Read the Paper, Write the Code: Agentic Reproduction of Social-Science Results

Anonymous ACL submission

Abstract

Recent work uses LLM agents to reproduce empirical social science results with access to both the data and code. We broaden this scope by asking: Can they reproduce results given only a paper's methods description and original data? We develop an agentic reproduction system that extracts structured methods descriptions from papers, runs reimplementations under strict information isolation—agents never see the original code, results, or paper—and enables deterministic, cell-level comparison of reproduced outputs to the original results. An LLM-based attribution step traces discrepancies through the system chain to identify root causes. Evaluating four agent scaffolds and four LLMs on 48 papers with human-verified reproducibility, we find that agents can largely recover published results, but performance varies substantially between models, scaffolds, and papers. Root cause analysis suggests that failures stem both from agent errors and from underspecification in the papers themselves.

1 Introduction

Agentic systems are increasingly able to autonomously generate, debug, and execute end-to-end research pipelines across a broad range of domains (Lu et al., 2024; Si et al., 2025; Gandhi et al., 2025; Schmidgall et al., 2025; Sun et al., 2025). These capabilities have led to the application of agents to the task of *scientific reproduction* (overview in Table 1), as well as to novel considerations of replication-oriented publication pipelines (Brodeur and Barbarioli, 2025; Fishman and Sekeres, 2025). In the empirical social sciences specifically, recent work uses agents to execute the pre-existing code associated with papers and to assess their level of reproducibility (Hu et al., 2025; Shah et al., 2026; Xu and Yang, 2026).

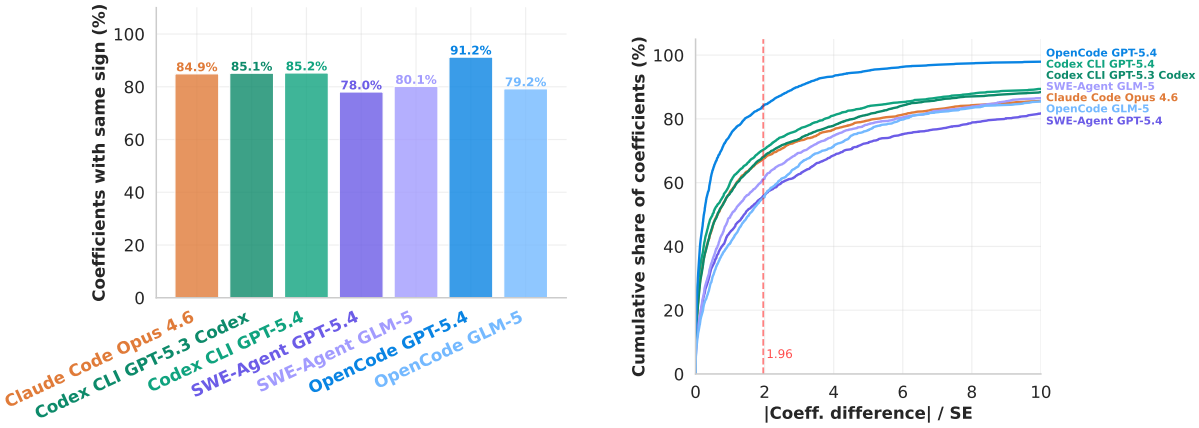
Yet publications, not code, communicate scientific research and serve as the source of truth. An open question, then, is whether reproduction of research results is contingent on direct access to the analysis code. Or whether, as presumably intended, the paper's methods description is sufficiently detailed to allow reproduction through re-implementation of the code from scratch.

This paper makes progress on this question by building an agentic system to reproduce research results given a paper's method description and associated data, but without the analysis code. Our pipeline extracts structured representations of the research methods and results, where specific numbers are masked. The agent is then tasked with reproducing the masked values by implementing the analysis pipeline from scratch, allowing for deterministic benchmarking.

We apply the pipeline to a dataset of 48 social science papers that have been human-verified as reproducible (Institute for Replication, 2024). Overall, frontier LLM agents can often re-implement analysis code of social science papers (Figure 1). For the three best-performing agents, reproduced and original coefficients agree on the sign over 85% of the time, and they are within the 95% confidence intervals over 70% of the time.

Performance varies substantially across LLMs and across agent scaffolds. The best-performing agent in our comparisons is GPT-5.4 using the OpenCode scaffold, out-performing Claude Code (Opus 4.6), GPT-5.3 Codex, and GLM-5 (Team et al., 2026). The scaffold–model interaction matters a lot, with GPT-5.4 on OpenCode outperforming GPT-5.4 on the Codex CLI or on mini-SWE-Agent. In supporting analysis, we show that OpenCode GPT-5.4's better performance is partly attributable to greater token usage, at the cost of more expensive API calls and longer analysis runs.

Our reproduction pipeline enables a detailed LLM-based diagnosis of failure modes to help identify the source of incorrect estimates. Errors mostly come from papers' underspecification of methods



(a) Share of reproduced coefficients with the same sign as the original paper.

(b) Cumulative dist. of |coefficient difference| / SE across approaches. Dashed red line marks 1.96 (95% CI threshold).

Figure 1: **Agentic systems can reliably recover a large share of the directional findings from social science publications given the publication text and data alone.** *Panel (a): The best performing agent recovers the correct sign over 90% of the time. Panel (b): The reproduced coefficients of the best performing agent are within the 95% confidence interval (based on the ground-truth standard error) over 80% of the time.*

and—to a lesser extent—from agents' misunderstanding of or noncompliance with the methods description.

These methods and findings expand on the recent literature on automated scientific reproducibility. While important gaps remain to be closed, LLM agents can increasingly *read the paper and write the code*. Such systems could therefore play an important role in addressing the well-recognized reproduction gaps in empirical social science (Ioannidis, 2005; Nosek et al., 2012; Brodeur et al., 2024).

2 Background on agentic reproducibility

Independent verification is central to the scientific method (Nosek et al., 2022). Recent crowdsourced reproduction efforts have revealed substantial variation in reproducibility across social science disciplines, but require considerable resources—hundreds of trained researchers to collectively assess only a negligible fraction of the published literature (Brodeur et al., 2026; Miske et al., 2026; Tyner et al., 2026).

A helpful framework from Dreber and Johannesson (2025) organizes different forms and components of reproducibility and replicability. *Reproducibility* tests whether the results and conclusions of original studies can be reproduced based on the *same data*. *Replicability* concerns the validity of results when similar methods are applied to *new* data. This paper concerns reproducibility.

Reproducibility can be further decomposed into three types: (1) re-running the original authors' code on identical data to regenerate results; (2) reproducing results using only the information reported in the paper and the same raw data; and (3) testing whether results are robust to reasonable alternative analytical decisions applied to the same data. Our work falls under (2), re-implementation.

System	Domain	Has Code	New Data	Eval.
CORE-Bench (Siegel et al., 2024)	Multi	✓		Det.
PaperBench (Starace et al., 2025)	ML			LLM
Paper2Code (Seo et al., 2026)	ML			LLM
AutoReproduce (Zhao et al., 2025)	ML			Both
FIRE-Bench (Wang et al., 2026)	ML			LLM
REPRO-Bench (Hu et al., 2025)	Soc. sci.	✓		LLM
PaperRepro (Zhang et al., 2026)	Soc. sci.	✓		LLM
Shah et al. (2026)	Soc. sci.	✓		Det.
Xu and Yang (2026)	Soc. sci.	✓		Det.
ReplicatorBench (Nguyen et al., 2026)	Soc. sci.	✓	✓	LLM
This work	**Soc. sci.**			**Det.**

Table 1: **Automated systems for scientific reproducibility.** *Columns indicate disciplinary domain (multidisciplinary, machine learning, or social science), whether the agent has access to the code, whether the agent replicates on new data, and the evaluation approach (deterministic, LLM-as-judge, or both).*

This paper contributes to ongoing research exploring the deployment of LLM agents for reproduction and replication, summarized in Table 1.[1] Existing benchmarks in the ML domain evaluate agents' ability to reproduce code architectures and results. In social science, prior works—Hu et al.

[1]We exclude benchmarks of code implementation quality rather than reproducibility (Huang et al., 2023; Tian et al., 2024; Hua et al., 2025).

(2025), Zhang et al. (2026), Shah et al. (2026), and Xu and Yang (2026)—benchmark reproducibility while granting agents access to the original code. Our contribution differs in that we reproduce social science results from the methods and data alone, without access to any author-provided code. In Nguyen et al. (2026), the agents try to replicate (rather than reproduce) a paper by retrieving new data and testing the same hypothesis.[2]

Unlike most prior work, which relies on LLM-as-judge evaluation, our evaluation is deterministic, comparing reproduced outputs directly to original values with adjustments for statistical significance based on ground-truth standard errors. This approach may be more reliable, since LLMs are not well-validated for evaluation tasks (Li et al., 2024).

3 A pipeline for paper-derived reproduction

We develop a multi-step pipeline to evaluate whether LLM agents can reproduce social science results. The pipeline tasks agents with re-implementing code from methods descriptions extracted from the paper, using the associated data (see Figure 2). Separating the pipeline into distinct steps minimizes information leakage of both existing numerical results and original code to the agents. The four steps are: (1) extracting data, methods, and results from the paper and associated files; (2) re-implementing the code based on the data, methods, and results table templates; (3) evaluating estimated results against the original results; and (4) diagnosing and explaining deviations.

3.1 Step 1: Extraction

The first step extracts structured information from the paper and its reproduction package in three sub-processes (exact prompts in Appendix M.1).

Method extraction. We provide an LLM with the complete PDF of the paper and prompt it to extract a complete methods description, similar to the approach taken by Seo et al. (2026) for machine learning papers. The methods description includes the main research question, the study setting (e.g., country and time period), data descriptions (e.g., data sources and a broad summary of each dataset), general data manipulations (e.g., filtering steps or variable construction), and detailed descriptions of each results table without the numerical contents (e.g., name, caption, row and column labels, table-specific data filters, and model specifications such as regression formulas). The model is explicitly instructed not to include any numerical results or conclusions, verified by checking for any numerals in the table description.

This step produces a structured representation of the paper's method without leaking results or code to the reimplementation agent (Section 3.2). Compared to providing the original paper directly (as is done in Nguyen et al. 2026), this step helps prevent the agent from hard-coding results or re-iterating code until exact results are reached. The quality of the methods extraction places a ceiling on downstream results, and incomplete or erroneous extractions are occasionally responsible for failed reproductions (Section 5.3).

Results extraction and blinding. The next step is to extract the original results from tables.[3] A dedicated LLM call retrieves the numerical values from all tables in the paper.[4] The model returns a structured representation of each table in which cells are identified by row and column indices and label names. For each cell, the model records the content as a string and decomposes it into a numerical value, a classification of the metric type (e.g., coefficient or number of observations), the number of significance stars, and—for statistics such as standard errors or p-values—a reference to the corresponding coefficient. To construct a template for the reproduction agents to fill in, we "blind" a copy of this structured table by setting all cell values and significance stars to null. Table parsing uses GPT-5-mini applied to the PDF rendered as images, which we found to outperform extraction from machine-readable PDF text. Manual validation on a random sample of five papers (24 tables) found perfect extraction for portrait-oriented tables and occasional minor numerical errors for landscape-oriented or multi-page tables.

[2] In one of Nguyen et al.'s (2026) specifications, the agent does not have access to the analysis code from the replicated paper, but it still has access to the complete original PDF and parsed results—the answers it aims to replicate. We remove such information in our approach because it can cause leakage—in preliminary versions of our setup where models had access to results, agents would often copy them directly.

[3] Our pipeline handles both tables and figures, but figure-level comparisons require vision-language models whose reliability for this task remains uncertain (Wang et al., 2024; Tang et al., 2025; Hou et al., 2026, *inter alia*). Supporting figure-level results are in Appendix B.

[4] We include all tables from the paper's published journal PDF. About 8% of tables (from 18 papers) are excluded due to extraction issues.

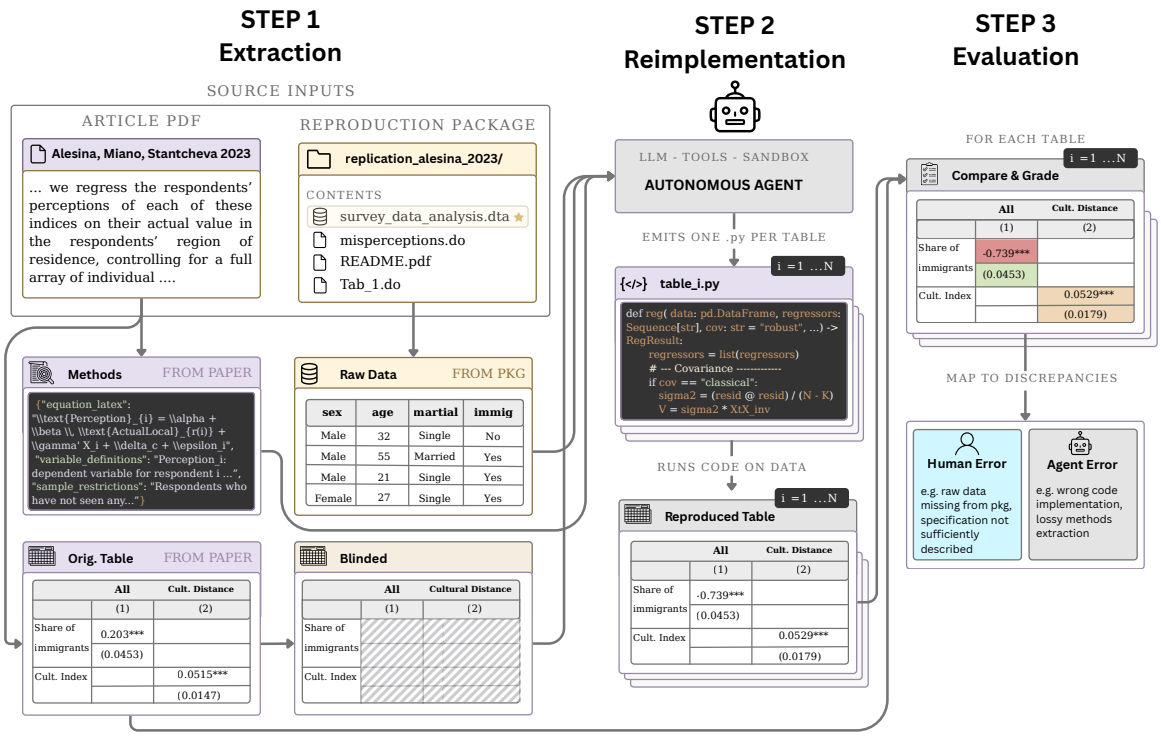


Figure 2: **Overview of the pipeline for replicating empirical results.** *Extraction parses paper + replication package; reimplementation runs an isolated agent that emits one Python script per table; evaluation compares cells against the original; explanation traces divergences to human or agent sources. See Section 3 for details.*

Data extraction. We instruct an LLM (GPT-5-mini) to identify the data files required for replication from the directory tree and README of each provided reproduction package, limited to the least pre-processed version available. The model also classifies whether the package contains sufficient data to fully reproduce the paper or whether data were partially or entirely missing (e.g., proprietary or subject to privacy restrictions). Papers with no usable data are skipped.

3.2 Step 2: Reimplementation

The extracted method, table templates, and original data serve as the sole inputs for the reproduction agents, following (Starace et al., 2025) and (Seo et al., 2026). Agents are instructed to read the extracted methods and table templates, explore the data, and write a separate Python script for each table that fills in the corresponding template (see Appendix M.2 for the prompt used).

The agents are restricted from accessing any file paths outside their workspace and are prohibited from retrieving the original reproduction package or paper via web tools. Each agent operates in an isolated environment containing only the task description, the extracted methods, the output templates, and a symlink to the data. The original reproduction package and paper PDF remain on the same machine but outside this boundary. These restrictions are necessary because agents could otherwise simplify the task by directly accessing the original code or results.

We implement a two-stage audit pipeline to verify compliance with the guardrail (Appendix J). The guardrail audit includes a deterministic regex scan, which classifies all accessed file paths and URLs as allowed or forbidden, as well as a GPT-5.4-mini review that rates each run from clean to severe violation. Suspicious runs are manually inspected and rerun if leakage is detected. In parallel, a hardcoding audit checks whether agents output statistical results as numeric literals without a computation path from the data. Together, these audits indicate that the reported results reflect genuine reimplementation rather than retrieval or memorization of pre-existing outputs.

3.3 Step 3: Evaluation

Both the agent-produced results and the original results share the same structured template, so we can directly compare outputs at the cell level. For all values, we check the sign agreement and compute

the percentage-point difference. For regression coefficients, we additionally compute the difference scaled by the ground-truth standard error.

Each numerical value receives a letter grade based on the percent absolute deviation from the original value. If the sign matches, we assign A (0%–2%), B (2%–20%), C (20%–40%), D (40%–60%), or E (>60%). If the sign doesn't match, we assign E. If the estimate is not produced or is not a number, we assign F (see Appendix C).

The letter grades have two advantages over the statistical scores. First, assigning E for >60% discrepancies is more robust to outlier estimates. Second, multiple criteria like the sign match can be combined into one grading scheme.

Results are aggregated first by computing the average grade across all cells within a table, and then by computing the average table grade within a paper. Binning into grades before aggregation improves interpretability and reduces the influence of outliers. Unlike some of the prior work, our approach avoids the ambiguity and opacity of LLM judges. In this regard, it is closest to (Xu and Yang, 2026), who also measure exact reproduction of coefficients.

3.4 Step 4: Explanation

The explanation step produces interpretable diagnostics of discrepancies between the original and reproduced results. An LLM agent (GPT-5.4 with Codex CLI) is prompted to perform the analysis as follows. The input is the set of observed cell-level numerical failures (all cells with a score of B or below). For each failing output, the agent is tasked with locating the relevant code in both the original reproduction package and the agent-generated Python code. It then identifies and describes the discrepancy causing the failure.

For each reported discrepancy, an LLM auditor runs consistency checks comparing the various input and output files. The auditor can identify four main types of errors. Under *Human Error*, we have *Missing Data* and *Paper vs. Code* (the original paper does not properly or completely describe the code, or contradicts it). Under *Agent Error*, we have *Paper vs. Methods Extraction* (the methods extraction pipeline does not properly describe the methods) or *Method Extraction vs. Agent* (the agent does not properly follow the extracted methods description).

4 Application to I4Replication data

Data Our analysis is based on papers with verified reproducibility. We draw on the work of *I4Replication*, an institute coordinating a large-scale research community effort to reproduce social science papers (Institute for Replication, 2024). Their process combines systematic review of reproduction packages, step-by-step code execution, and direct exchange with the original authors, resulting in a detailed reproducibility report and classification for each paper. We collect the PDFs and reproduction packages of all papers that I4Replication classifies as fully reproducible, leading to a total dataset of 48 papers (Appendix Table 2).

Appendix Table A lists all the papers with associated journal, title, main programming language, and number of lines of code. Table 3 Panel A tabulates the counts by journal. The papers are from a mix of well-regarded economics and political science journals. Summary stats on the code are shown in Appendix Figure 22. The original analysis code is mostly programmed in Stata (54%) or R (27.1%), with zero papers in Python. The packages have an average of 5,324 lines of code.

Summary statistics on the extracted paper elements are reported in Table 3 Panel B. The system extracted 222 ground-truth tables containing 14,214 table cells. Of these, about one-third (5,149) contain regression coefficients. About four-fifths (4,253) of coefficients are accompanied by a standard error, while one-sixth (779) have p-values. Other frequently observed statistics are the number of observations and R-squared.

A potential issue with this application is data leakage: tested LLMs may have been exposed to the manuscripts or code during pre-training. To probe the relevance of leakage, we provide an auxiliary analysis reproducing results from papers published after the model knowledge cutoff, which is August 2025 for both GPT 5.3-Codex/5.4 (OpenAI, 2026) and Claude Opus 4.6 (Anthropic, 2026). Specifically, we sample ten papers (with reproduction packages) from the *Economic Journal* (EJ), with five papers published *before* and five papers published *after* August 2025. While these papers have not yet been verified as reproducible by I4Replication, they have been verified by EJ's data editor team, which verifies that reproduction packages produce the same outputs as the paper.

Agent systems The LLM agents tested are GPT-5.4, GPT-5.3 Codex, Claude Opus 4.6, and the

open-weights GLM-5 (Team et al., 2026). We evaluate two proprietary agent scaffolds—Claude Code and Codex—and two open-source alternatives—mini-SWE-agent (Yang et al., 2024) and OpenCode.[5] For the open-source scaffolds, we compare a proprietary model (GPT-5.4) and an open-weights alternative (GLM-5). Figure 25 illustrates some sample code segments generated by the agents.

Before evaluating success in reproduction, a preliminary question is whether these agent systems can even generate any results at all. Table 3 Panel C summarizes the completion rates according to various features of the paper results. The agents deliver usable results for at least 92% (and up to 100%) of papers, 82% (up to 97%) of tables, and 52% (up to 72%) of cells. The bottom part of the table shows completion across various types of cells, with the most important ones being coefficients (82% completion on average) and standard errors (80% completion on average). In terms of overall completion, the best agent system is Claude Opus 4.6 and the worst is SWE-Agent GLM-5.

5 Results: How well do AI agents reproduce papers without their code?

5.1 Main results

This section presents the main results on reproducing the I4Replication papers. We start at the cell (estimate) level and proceed to table- and paper-level results.

Cell-level. Figure 1 reports cell-level results for regression coefficients, which are the most prevalent statistic in tables. First, Figure 1a reports the share of reproduced estimates that match the sign of the original estimates. Across all models and scaffolds, reproduced coefficients match the sign for the large majority of observations, ranging from 78% (SWE-Agent GPT-5.4) to 91% (OpenCode GPT-5.4). In all cases this is a significant lift over the naive "guess positive," which would give 68% accuracy. The numbers in Figure 1a ignore "missing" coefficients that were not generated; Appendix Figure 8 includes them in the denominator.

Figure 1b presents cumulative distributions (CDFs) of the absolute difference of the original and reproduced coefficients, divided by the ground-truth standard error. This normalized difference measures the discrepancy between original and reproduced coefficients in statistically meaningful units (the same units as the Wald t-statistic). For these curves, better performance is indicated by hugging the top-left corner. The value of the CDF at $x = 1.96$ indicates the share of reproduced estimates which lie within the 95% confidence interval (CI), meaning they are not statistically different from the original estimates. The reproduced coefficient falls within the 95% CI for more than half of all reproduced values, even in the worst case. For the best model (OpenCode GPT-5.4), over 80% of reproduced values are within the 95% CI.

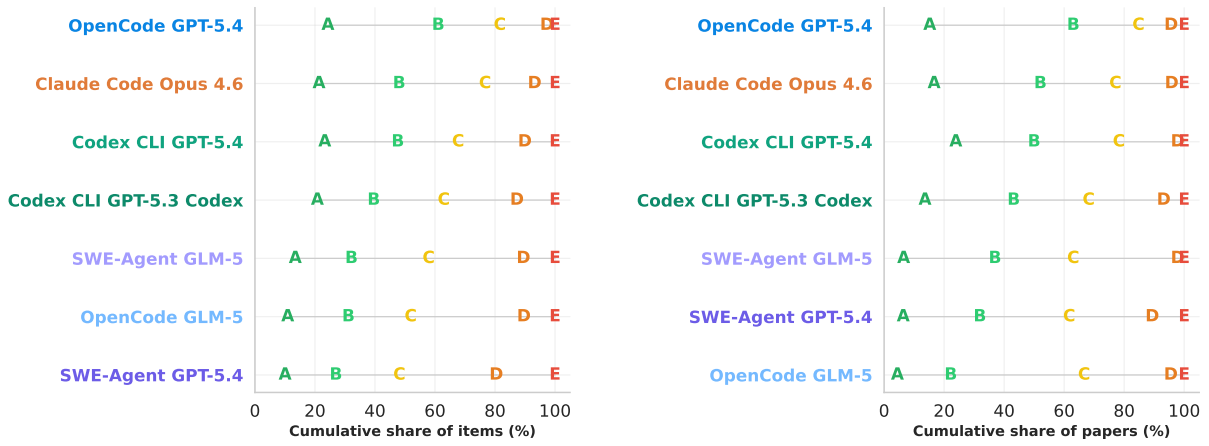
The CDFs further reveal that a non-negligible share of reproduced coefficients deviate substantially from the originals. An informal inspection of these cases suggests that while some are attributable to agent behavior (e.g., selecting the wrong dataset from among several similar candidates), the majority stem from unit scaling issues not accounted for by our comparison system (e.g., the original authors report values in dollars, whereas the dataset is denominated in cents).

Appendix Figure 7 extends this analysis with the absolute percentage-point difference between original and reproduced statistics broken out by type (coefficient, standard error, R-squared, and p-value). Across all statistic types, the best agent is OpenCode GPT-5.4. The worst agent is SWE-Agent GPT-5.4 (3/4 statistics) or OpenCode GLM-5 (1/4 statistics). Codex GPT-5.4, Codex GPT-5.3, and Claude Code Opus 4.6 perform similarly, forming a second tier behind OpenCode GPT-5.4. SWE-Agent performs the worst among the scaffolds, even with GPT-5.4 as the underlying model.

Table- and paper-level. Figure 3 reports results with letter grades, aggregated by table (panel a) and paper (panel b). Converting numerical values to grades mitigates the influence of outliers when comparing and aggregating results. Overall, the results are consistent when looking at tables and papers. The table-level aggregation highlights performance differences across agents: the best-performing model–scaffold combination reproduces twice as many tables near-perfectly (grade A) compared to the worst-performing one.

Both the table- and paper-level aggregations indicate that achieving accurate and reliable reproduction beyond individual cells remains challenging for all agents. Agent rankings become less differentiated at the paper level, suggesting that some agents tend to perform consistently well across all tables once they successfully reproduce one (e.g.,

[5] https://opencode.ai/



(a) Table-level grades

(b) Paper-level grades

Figure 3: **Grade comparison by aggregation level and agent.** *Grades are assigned to percentage differences between reproduced and original results following the rubric in Table 6 and aggregated with equal weights. All reproduced numbers are rounded to the number of digits reported in the original papers. The best scaffolds and models reproduce more than a fifth of tables almost exactly (average cell within 2%, an A grade), and more than 60% within 20% (a B); comparable, but slightly smaller proportions hold at the level of the entire paper. Observations are ranked by their share of A and B grades. Equivalent figures including F-grades are in Appendix Fig. 6.*

Codex with GPT-5.4), whereas others solve individual tables more independently, leading to greater within-paper performance variance.

Appendix Figure 11 shows table results separately by the empirical function of the table—main results, mechanism analysis, robustness checks, or descriptive statistics. Descriptive statistics are easier to reproduce than results tables, which themselves show similar performance across main, mechanisms, and robustness categories. Appendix H provides other supporting results on correlates of reproduction performance.

5.2 Variation in agentic effort

Performance differences across agents partly reflect differences in computational effort. The best-performing agent, OpenCode GPT-5.4, consumes substantially more tokens, takes much more time per run, and is the most costly configuration. Lower-performing agents operate under tighter compute budgets that may limit their ability to resolve ambiguities or recover from intermediate errors. Prior work emphasizes the importance of scaffold–model interaction for agentic performance (Cohn et al., 2026); our findings also suggest that differences in observed accuracy may partly reflect differences in implicit compute budgets rather than purely differences in capability. (Token/runtime/-costs breakdowns and a tool-call-level analysis of effort in App. E).

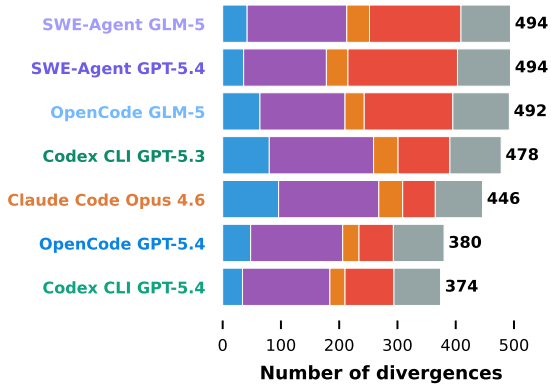


Figure 4: **Error analysis pipeline: number of divergences by error source.** *The error pipeline identifies divergences between the original and reproduced code for tables that score worse than overall grade B and traces the root cause. Categories: **agent error** (the agent mismaps methods to code), **extractor error** (faulty paper-to-methods extraction), **original error** (paper under- or mis-specifies the original code), **data missing**, and **other / unknown**. Appendix I has more detail.*

5.3 Exploration of error sources

Where do these errors originate? Figure 4 summarizes the root causes identified by our discrepancy pipeline across model–scaffold combinations, grouped into five categories: *agent error*, *extractor error*, *original error*, *data missing*, and *other/unknown*. As shown, the large majority of divergences—over three-fourths—can be traced to

a specific, interpretable source.

A substantial share of discrepancies are outside the reproducing agent's control. The largest share of divergences stems from *original errors*, indicating mismatches between the paper and the underlying code; in other words, the methods descriptions are often insufficiently precise to enable faithful reimplementation. Another large share of errors come from missing data. Finally, some discrepancies arise from errors in the shared extraction pipeline, before the agent systems start their work.

Agent errors constitute the second largest category of discrepancies. For the strongest agents, the share of agent errors declines markedly, becoming a relatively minor component of total discrepancies. A larger relative fraction of divergences is then attributed to human sources—either to underspecification in the original paper or to missing data. This pattern suggests that limitations in how methods are documented, rather than agent capability alone, are a primary barrier to automated reproducibility.

There is substantial variation across agents in the number of discrepancies attributed to human sources, such as paper underspecification. This variation does not imply that some agents encounter fewer human-caused issues, but rather that they handle these ambiguities differently. Even when the underlying discrepancy originates from incomplete or unclear reporting in the paper, agents may still reproduce the correct result if their implicit assumptions happen to align with the original implementation. As a result, the observed number of human-attributed discrepancies reflects both the presence of underspecification and the agent's ability to resolve it correctly. Appendix Table 8 illustrates this mechanism with three concrete examples in which different agents make different default assumptions when the paper omits a key detail.

5.4 Robustness to repeated runs and pre-training leakage

Re-run stability (appendix F). Rerunning the full pipeline twice for 20 randomly selected papers using Claude Code and Codex GPT-5.4 yields three runs per paper. More than 80% of tables exhibit a grade spread of at most one across runs, indicating that most reproduction outcomes are robust to randomness. At the table level, stability masks variation at the coefficient level: about half of estimated coefficients are statistically different from themselves (same cell across runs); interpret fine-grained between-agent comparisons with caution.

Leakage analysis (appendix G). To assess the potential role of pre-training leakage, we reproduce five *EJ* papers published before and five published after the model knowledge cutoff (August 2025) using Claude Code (Opus 4.6) and Codex (GPT-5.4). For both agents, we find no statistically significant difference in performance between pre- and post-cutoff papers; if anything, post-cutoff performance is slightly higher. Mechanistically, such leakage would also be difficult to exploit: any memorized results would need translation into executable Python from packages in other languages.

6 Conclusion

Across a diverse set of empirical papers, agents reconstruct a large share of published results from a paper's text and data alone, demonstrating substantial progress toward automated reproducibility. Overall failure modes indicate that, going forward, the primary bottleneck for further progress may not be model capability but the way social science methods are documented: key implementation details are often underspecified, forcing agents to infer choices just as a human researcher would. As a result, the ability to reproduce results reflects a combination of both the agent's capabilities and the paper's clarity and completeness.

Our findings also speak to the value of paper-derived reproduction even when code is available. Re-running provided code only certifies that the code produces the reported numbers; it cannot reveal whether the paper, on its own terms, describes a procedure that yields those numbers. This gap is clear in our setup, where substantial share of failures come from *paper–code* mismatches that code-only verification would not surface. This suggests a clearer division of roles: code as the authoritative representation of *what was done*, and the paper as the explanation of *why*.

Rather than expecting papers to fully encode implementation details, a more robust approach is to require explicit alignment between the narrative and the underlying code. Here, agentic systems are a natural fit: translating papers into executable code, comparing outputs, and surfacing inconsistencies between the two representations. Thus, automated reproduction can serve as both a tool for verification and a diagnostic, identifying where methods are underspecified, where implementations diverge, and where assumptions must be inferred.

Limitations

Our study design comes with several limitations. First, our error attribution pipeline relies on LLM-based judgements over multi-file, multi-language reproduction packages; while we manually validate components of the system, the attribution itself is not independently verified, and the precise share of agent-vs-human-attributable errors should be interpreted as a qualitative signal rather than a precise estimate. Second, the methods extractor sits upstream of the reproduction agent, so its quality places a ceiling on overall performance. Future work should compare alternative extraction pipelines to disentangle extractor quality from agent capability. Third, while our pre/post cutoff analysis suggests pre-training leakage is not the main driver of performance, the sample is small and limited to one journal, and post-training data contamination cannot be ruled out. Fourth, results from agentic systems are stochastic; we report a re-run stability analysis for two of the agents (Section 5.4), but a systematic within-agent consistency analysis across all model–scaffold combinations was not feasible due to cost. Finally, our sample of 48 papers is restricted to studies that I4Replication has verified as fully reproducible. While the papers cover multiple disciplines and a range of topics and paper formats, the paper selection process is non-random, so the sample need not represent empirical social science more broadly.

We also acknowledge that our work only covers one step of the reproducibility ladder, termed "recreate reproducibility" by Dreber and Johannesson. Assessing how agents perform on tasks at a higher level of abstraction—e.g., testing robustness to other "reasonable analytical decisions"—remains an open question for future work.

Ethics Statement

All papers and reproduction materials are already made public via I4Replication; we do not redistribute the original work (papers or original reproduction packages). As we describe, failure of agentic reproduction can have multiple causes. Therefore, neither success nor failure of agentic reproduction should be interpreted as a quality signal of the reproduced work.

Last, moving beyond reproduction with pre-existing data and known methods raises a series of questions about the role and implications of agentic assistance in scientific work. What if the data are unavailable, requiring agents to recover or reconstruct them? What if only the research question or hypothesis is given, and methods must be inferred rather than followed? What if agents are tasked not just with reproducing results, but with refining the analysis through specification checks, falsification of identification assumptions, or exploration of underlying mechanisms? And what if the goal shifts from reproduction to replication — asking questions or applying methods with new data?

While tackling these extensions could help address ongoing concerns about reproducibility in empirical research (e.g. Brodeur et al., 2024, 2026), such progress would also involve pushing agentic systems further toward active participation in the research process. This shift requires us to define new criteria for scientific validity, including how to assess identification, robustness, and the reliability of conclusions generated without direct human oversight.

References

Anthropic. 2026. Models overview. Anthropic API Documentation. Accessed: 2026-04-01.

Abel Brodeur and Bruno Barbarioli. 2025. The replication engine. Institute for Progress.

Abel Brodeur, Derek Mikola, Nikolai Cook, Thomas Brailey, Ryan Briggs, Alexandra de Gendre, Yannick Dupraz, Lenka Fiala, Jacopo Gabani, Romain Gauriot, Joanne Haddad, Ryan McWay, Joel Levin, Magnus Johannesson, Edward Miguel, Lennard Metson, Jonas Minet Kinge, Wenjie Tian, Timo Wochner, and 30 others. 2024. Mass reproducibility and replicability: A new hope. Technical Report 107, Institute for Replication (I4R).

Abel Brodeur, Derek Mikola, Nikolai Cook, Lenka Fiala, Thomas Brailey, Ryan Briggs, Alexandra de Gendre, Yannick Dupraz, Jacopo Gabani, Romain Gauriot, Joanne Haddad, Goncalo Lima, Jörg Ankel-Peters, Anna Dreber, Douglas Campbell, Lamis Kattan, Diego Marino Fages, Fabian Mierisch, Pu Sun, and 328 others. 2026. Reproducibility and robustness of economics and political science research. *Nature*, 652(8108):151–156.

Clayton Cohn, Siyuan Guo, Surya Rayala, Hanchen David Wang, Naveeduddin Mohammed, Umesh Timalsina, Shruti Jain, Angela Eeds, Menton Deweese, Pamela J. Osborn Popp, Rebekah Stanton, Shakeera Walker, Meiyi Ma, and Gautam Biswas. 2026. Evidence-decision-feedback: Theory-driven adaptive scaffolding for llm agents. *Preprint*, arXiv:2602.01415.

Anna Dreber and Magnus Johannesson. 2025. A framework for evaluating reproducibility and replicability in economics. *Economic Inquiry*, 63(2):338–356.

Nic Fishman and Gabriel Sekeres. 2025. Editorial screening when science is cheap.

Shubham Gandhi, Dhruv Shah, Manasi Patwardhan, Lovekesh Vig, and Gautam Shroff. 2025. Researchcodeagent: An llm multi-agent system for automated codification of research methodologies. In *AI for Research and Scalable, Efficient Systems*, pages 3–37, Singapore. Springer Nature Singapore.

Yifan Hou, Buse Giledereli, Yilei Tu, and Mrinmaya Sachan. 2026. Do vision-language models really understand visual language? In *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25. JMLR.org.

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*, pages 23616–23626, Vienna, Austria. Association for Computational Linguistics.

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. *Preprint*, arXiv:2506.02314.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2023. MLAgentBench: Evaluating language agents on machine learning experimentation. *Preprint*, arXiv:2310.03302.

Institute for Replication. 2024. Meta database, version 1. `https://i4replication.org/reports/`. Accessed: 2026-03-31.

John P. A. Ioannidis. 2005. Why most published research findings are false. *PLoS Medicine*, 2(8):e124.

Haitao Li, Qian Dong, Junjie Chen, Huixue Su, Yujia Zhou, Qingyao Ai, Ziyi Ye, and Yiqun Liu. 2024. Llms-as-judges: a comprehensive survey on llm-based evaluation methods. *arXiv preprint arXiv:2412.05579*.

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

Olivia Miske, Anna Lou Abatayo, Mason Daley, Mirka Dirzo, Nicholas Fox, Noah Haber, Krystal M Hahn, Melissa Kline Struhl, Brinna Mawhinney, Priya Silverstein, Theresa Stankov, Andrew H Tyner, Matúš Adamkovič, Shilaan Alzahawi, Saule Anafinova, Eli Awtrey, Erick Axxe, James Bailey, Bert N Bakker, and 109 others. 2026. Investigating the reproducibility of the social and behavioural sciences. *Nature*, 652(8108):126–134.

Bang Nguyen, Dominik Soós, Qian Ma, Rochana R. Obadage, Zack Ranjan, Sai Koneru, Timothy M. Errington, Shakhlo Nematova, Sarah Rajtmajer, Jian Wu, and Meng Jiang. 2026. ReplicatorBench: Benchmarking LLM agents for replicability in social and behavioral sciences. *Preprint*, arXiv:2602.11354.

Brian A. Nosek, Tom E. Hardwicke, Hannah Moshontz, Aurélien Allard, Katherine S. Corker, Anna Dreber, Fiona Fidler, Joseph Hilgard, Melissa Kline Struhl, Michèle B. Nuijten, Julia M. Rohrer, Felipe Romero, Anne M. Scheel, Laura D. Scherer, Felix D. Schönbrodt, and Simine Vazire. 2022. Replicability, robustness, and reproducibility in psychological science. *Annual Review of Psychology*, 73:719–748.

Brian A. Nosek and 1 others. 2012. An open, large-scale, collaborative effort to estimate the reproducibility of psychological science. *Perspectives on Psychological Science*, 7(6):657–660.

OpenAI. 2026. GPT-5.4 pro model. OpenAI API Documentation. Accessed: 2026-04-01.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. 2025. Agent laboratory: Using LLM agents as research assistants. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 5977–6043, Suzhou, China. Association for Computational Linguistics.

Minju Seo, Jinheon Baek, Seongyun Lee, and Sung Ju Hwang. 2026. Paper2Code: Automating code generation from scientific papers in machine learning. In *International Conference on Learning Representations (ICLR)*.

Syed Mehtab Hussain Shah, Frank Hopfgartner, and Arnim Bleier. 2026. Automating computational reproducibility in social science: Comparing prompt-based and agent-based approaches. *Preprint*, arXiv:2602.08561.

Chenglei Si, Diyi Yang, and Tatsunori Hashimoto. 2025. Can LLMs generate novel research ideas? A large-scale human study with 100+ NLP researchers. In *International Conference on Learning Representations (ICLR)*.

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

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

Maojun Sun, Ruijian Han, Binyan Jiang, Houduo Qi, Defeng Sun, Yancheng Yuan, and Jian Huang. 2025. A survey on large language model-based agents for statistics and data science. *The American Statistician*, 0(0):1–14.

Liyan Tang, Grace Kim, Xinyu Zhao, Thom Lake, Wenxuan Ding, Fangcong Yin, Prasann Singhal, Manya Wadhwa, Zeyu Leo Liu, Zayne Rea Sprague, Ramya Namuduri, Bodun Hu, Juan Diego Rodriguez, Puyuan Peng, and Greg Durrett. 2025. Chartmuseum: Testing visual reasoning capabilities of large vision-language models. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*.

GLM-5 Team, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chengxing Xie, and 1 others. 2026. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*.

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, and 11 others. 2024. SciCode: A research coding benchmark curated by scientists. *Preprint*, arXiv:2407.13168.

Andrew H Tyner, Anna Lou Abatayo, Mason Daley, Samuel Field, Nicholas Fox, Noah A Haber, Krystal M Hahn, Melissa Kline Struhl, Brinna Mawhinney, Olivia Miske, Priya Silverstein, Courtney K Soderberg, Theresa Stankov, Ahmed Abbasi, Christopher L Aberson, Balazs Aczel, Matúš Adamkovič, Nihan Albayrak, Peter J Allen, and 273 others. 2026. Investigating the replicability of the social and behavioural sciences. *Nature*, 652(8108):143–150.

Zhen Wang, Fan Bai, Zhongyan Luo, Jinyan Su, Kaiser Sun, Xinle Yu, Jieyuan Liu, Kun Zhou, Claire Cardie, Mark Dredze, Eric P. Xing, and Zhiting Hu. 2026. FIRE-Bench: Evaluating agents on the rediscovery of scientific insights. *Preprint*, arXiv:2602.02905.

Zirui Wang, Mengzhou Xia, Luxi He, Howard Chen, Yitao Liu, Richard Zhu, Kaiqu Liang, Xindi Wu, Haotian Liu, Sadhika Malladi, Alexis Chevalier, Sanjeev Arora, and Danqi Chen. 2024. Charxiv: Charting gaps in realistic chart understanding in multimodal LLMs. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*.

Yiqing Xu and Leo Yang Yang. 2026. Scaling reproducibility: An AI-assisted workflow for large-scale replication and reanalysis. *Preprint*, arXiv:2602.16733.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Linhao Zhang, Tong Xia, Jinghua Piao, Lizhen Cui, and Yong Li. 2026. PaperRepro: Automated computational reproducibility assessment for social science papers. *Preprint*, arXiv:2603.00058.

Xuanle Zhao, Zilin Sang, Yuxuan Li, Qi Shi, Weilun Zhao, Shuo Wang, Duzhen Zhang, Xu Han, Zhiyuan Liu, and Maosong Sun. 2025. AutoReproduce: Automatic AI experiment reproduction with paper lineage. *Preprint*, arXiv:2505.20662.

A Data overview

Step	N
I4R Replicate universe (unique papers)	109
Perfect computational reproduction	59
Sufficient or partial data availability	54
Has extractable tables	48
Final sample	48

Table 2: **Sample Construction.** Decomposition of papers from I4R.

Panel A: Papers by journal and discipline (final sample, $N = 48$)

Journal	Discipline	N papers
American Journal of Political Science (AJPS)	Political Science	11
Economic Journal (EJ)	Economics	9
American Economic Review (AER)	Economics	8
American Economic Journal: Economic Policy (AEJ:Pol)	Economics	5
Journal of Politics (JOP)	Political Science	5
American Economic Journal: Applied Economics (AEJ:AE)	Economics	3
American Political Science Review (APSR)	Political Science	3
Review of Economic Studies (REStud)	Economics	2
American Economic Journal: Macroeconomics (AEJ:Macro)	Economics	1
Quarterly Journal of Economics (QJE)	Economics	1
Total		48

Panel B: Extracted Elements in Original Papers (per paper, $N = 48$)

	Total	Mean	SD	Min	Max
Tables	**222**	4.6	2.8	1	11
Cells (present)	**14,214**	296.1	261.4	20	1433
Coefficients	**5,149**	107.3	93.5	8	384
Standard errors	**4,253**	88.6	85.3	0	319
p-values	**779**	16.2	32.5	0	127
t-statistics	**10**	0.2	1.4	0	10
Confidence intervals	**112**	2.3	16.2	0	112
R-squared	**590**	12.3	17.9	0	58
N observations	**1,701**	35.4	86.2	0	602
F-statistics	**121**	2.5	6.2	0	30
Other numeric	**1,607**	33.5	53.6	0	253

Panel C: Completion Counts and Rates (%) by Paper Element and Agent System

Scaffold	**Claude Code**	**Codex CLI**		**SWE-Agent**		**OpenCode**		**Average**
LLM	**Opus 4.6**	**GPT-5.3 Codex**	**GPT-5.4**	**GPT-5.4**	**GLM-5**	**GPT-5.4**	**GLM-5**	
Papers	48 (100%)	45 (94%)	45 (94%)	46 (96%)	45 (94%)	45 (94%)	44 (92%)	**95%**
Tables	213 (96%)	215 (97%)	214 (96%)	204 (92%)	182 (82%)	207 (93%)	213 (96%)	**95%**
Cells (Overall)	**10,276 (72%)**	**9,566 (67%)**	**9,615 (68%)**	**8,101 (57%)**	**7,374 (52%)**	**9,096 (64%)**	**7,648 (54%)**	**62%**
Coefficients	4,355 (85%)	4,315 (84%)	4,395 (85%)	4,113 (80%)	3,907 (76%)	4,180 (81%)	4,326 (84%)	**82%**
Standard errors	3,414 (80%)	3,623 (85%)	3,541 (83%)	3,257 (77%)	2,955 (69%)	3,458 (81%)	3,443 (81%)	**80%**
p-values	692 (89%)	590 (76%)	697 (89%)	617 (79%)	663 (85%)	501 (64%)	655 (84%)	**81%**
t-statistics	10 (100%)	10 (100%)	10 (100%)	10 (100%)	10 (100%)	10 (100%)	10 (100%)	**100%**
Confidence intervals	58 (52%)	54 (48%)	54 (48%)	54 (48%)	54 (48%)	54 (48%)	54 (48%)	**49%**
R-squared	498 (84%)	524 (89%)	488 (83%)	506 (86%)	424 (72%)	428 (73%)	502 (85%)	**82%**
N observations	1,267 (74%)	1,402 (82%)	1,397 (82%)	1,315 (77%)	858 (50%)	1,341 (79%)	1,383 (81%)	**75%**
F-statistics	108 (89%)	109 (90%)	105 (87%)	111 (92%)	62 (51%)	109 (90%)	111 (92%)	**84%**
Other numeric	1,175 (73%)	1,326 (83%)	1,248 (78%)	1,261 (78%)	1,036 (64%)	1,296 (81%)	1,240 (77%)	**76%**

Table 3: **Descriptive overview.** *Panel A: number of included papers by journal. Panel B: per-paper summary statistics on the original outputs. Panel C: reproductions, cell counts given as absolute totals with the percentage of originals in parentheses; Average is the mean of the shares across the 7 approach–model combinations.*

Table 4: Papers in Final Sample

Journal	Title	Year	Language	LOC
APSR	Flight to Safety: COVID-Induced Changes in the Intensity of Status Quo Preference and Voting Behavior	2021	R	2,372
APSR	Campaign Contributions and Roll-Call Voting in the U.S. House of Representatives: The Case of the Sugar Industry	2022	Stata	93
APSR	Can't We All Just Get Along? How Women MPs Can Ameliorate Affective Polarization in Western Publics	2022	R	716
JOP	Not All Elections Are Created Equal: Election Quality and Civil Conflict	2021	Stata	148
JOP	Antinormative Messaging, Group Cues, and the Nuclear Ban Treaty	2021	R	1,544
JOP	Changing Tides: Public Attitudes on Climate Migration	2021	R	3,720
JOP	What Makes Anticorruption Punishment Popular? Individual-Level Evidence from China	2021	R	1,328
JOP	Black Workers in White Places: Daytime Racial Diversity and White Public Opinion	2021	R	1,297
EJ	Non-Linearities, State-Dependent Prices and the Transmission Mechanism of Monetary Policy	2021	MATLAB	5,520
EJ	Understanding Ethnolinguistic Differences: The Roles of Geography and Trade	2021	Stata	1,362
EJ	Gender Differences in Cooperative Environments? Evidence from the U.S. Congress	2021	Stata	16,514
EJ	Pre-Colonial Warfare and Long-Run Development in India	2021	Stata	4,758
EJ	Spillover Effects of Intellectual Property Protection in the Interwar Aircraft Industry	2021	Stata	2,595
EJ	Why Don't Firms Hire Young Workers During Recessions?	2021	Stata	1,127
EJ	The Wheels of Change: Technology Adoption, Millwrights and the Persistence in Britain's Industrialisation	2022	Stata	1,240
EJ	Peer Effects in Academic Research: Senders and Receivers	2022	mixed	5,606
	The Power of Hydroelectric Dams: Historical Evidence from the United States over the Twentieth Century	2022	Stata	7,598
QJE	War, Socialism, and the Rise of Fascism: an Empirical Exploration	2022	Stata	2,415
REStud	Who Chooses Commitment? Evidence and Welfare Implications	2021	mixed	4,823
	Exposure and Preferences: Evidence from Indian Slums	2020	R	2,399
AJPS	Re-Assessing Elite-Public Gaps in Political Behavior	2020	R	1,731
AJPS	Public Infrastructure and Economic Development: Evidence from Postal Systems	2021	mixed	1,074
AJPS	Hate Crimes and Gender Imbalances: Fears over Mate Competition and Violence against Refugees	2021	R	3,068
AJPS	Ascriptive Characteristics and Perceptions of Impropriety in the Rule of Law: Race, Gender, and Public Assessments of Whether Judges Can Be Impartial	2021	R	2,118
AJPS	Decentralization Can Increase Cooperation Among Public Officials	2021	mixed	1,028
AJPS	The Geography of Repression and Opposition to Autocracy	2021	Stata	8,345
AJPS	Talking Shops: The Effects of Caucus Discussion on Policy Coalitions	2021	R	890
AJPS	Parties as Disciplinarians: Charisma and Commitment Problems in Programmatic Campaigning	2021	mixed	1,445
AJPS	Indecent Disclosures: Anticorruption Reforms and Political Selection	2021	R	1,075
REStud	Immigration and Redistribution	2023	mixed	5,287
AER	Interaction, Stereotypes, and Performance: Evidence from South Africa	2022	Stata	12,428
AER	Vulnerability and Clientelism	2022	Stata	5,412
AER	Enabling or Limiting Cognitive Flexibility? Evidence of Demand for Moral Commitment	2023	Stata	4,656
AER	Market Access and Quality Upgrading: Evidence from Four Field Experiments	2022	Stata	3,828
AER	Evaluating Deliberative Competence: A Simple Method with an Application to Financial Choice	2022	Stata	3,178
AER	When a Doctor Falls from the Sky: The Impact of Easing Doctor Supply Constraints on Mortality	2023	Stata	5,230
AER	Can Technology Solve the Principal-Agent Problem? Evidence from China's War on Air Pollution	2022	Stata	1,864
AER: Insights	Wage Cyclicality and Labor Market Sorting	2022	Stata	16,880

Journal	Title	Year	Language	LOC
AEJ: Applied	Assortative Matching at the Top of the Distribution: Evidence from the World's Most Exclusive Marriage Market	2022	Stata	12,957
AEJ: Applied	Historical Lynchings and the Contemporary Voting Behavior of Blacks	2022	Stata	1,432
	How Effective Are Monetary Incentives to Vote? Evidence from a Nationwide Policy	2021	Stata	7,523
AEJ: Macro	Declining Worker Turnover: The Role of Short-Duration Employment Spells	2021	mixed	2,374
AEJ: Policy	Multinationals' Sales and Profit Shifting in Tax Havens	2022	Stata	7,081
AEJ: Policy	School Spending and Student Outcomes: Evidence from Revenue Limit Elections in Wisconsin	2022	Stata	10,813
AEJ: Policy	The Long-Run Effects of Sports Club Vouchers for Primary School Children	2022	Stata	29,492
AEJ: Policy	How Do Beliefs about the Gender Wage Gap Affect the Demand for Public Policy?	2022	Stata	34,984
AJPS	Entertaining Beliefs in Economic Mobility	2022	R	2,204

Step	Name	Inputs	Outputs
1	Extraction	Paper PDF Reproduction Package	Raw Data Extracted Methods Original Table Table Template
2	Reimplementation	Raw Data Extracted Methods Table Template	Script$_i$.py Table$_i$
3	Evaluation	Table$_i$ Original Table	Comparison
4	Explanation	Paper PDF Reproduction Package Extracted Methods Script$_i$.py Comparison	Discrepancies & Error Sources

Table 5: Inputs and outputs for each step of the reproduction pipeline.

B Reproduction of Figures

Our pipeline handles both tables and figures as reproduced outputs. For figures, the extractor generates a detailed description of the axes and plot structure without referencing any underlying data or results, and produces a simple Python template that recreates the plot and its styling from a data object (e.g., a dataframe). This design allows the reimplementation agent to focus on reconstructing the data while avoiding exposure to the original plot and minimizing the risk of result leakage, while maintaining stylistic consistency across runs.

Evaluating reproduced figures is inherently more complex than comparing tables, as it requires assessing overall trends, individual data points, axes, and other visual features. Automating this process therefore relies on visual language models. However, whether such models can provide fair and consistent evaluations remains an open question. For this reason, we do not report quantitative results for figures. Figure 5 illustrates a representative example from GPT-5.4 in OpenCode, demonstrating the feasibility of the approach: the reproduced figures capture the main trends of the originals, though some visual discrepancies remain.

C Details on A–F grading

Tables are evaluated through a comparison of aligned original and replicated outputs. First, cells are aligned across the two tables and replicated values may be rescaled when the comparison detects an approximate power-of-ten mismatch. Grades are assigned from the resulting cell-level differences. Because percentage differences are unstable when the original value is close to zero, the grading rule distinguishes between near-zero and non-near-zero cases. The grading rubric is outlined in Table 6.

Table-level grades are computed by converting cell grades to numeric values ($A = 5$, $B = 4$, $C = 3$, $D = 2$, $E = 1$, $F = 0$), averaging over non-F cells only, and then mapping the average back to a letter grade using fixed thresholds: $[4.5, 5] \rightarrow A$, $[3.5, 4.5) \rightarrow B$, $[2.5, 3.5) \rightarrow C$, $[1.5, 2.5) \rightarrow D$, and $[0.5, 1.5) \rightarrow E$. If all cells are graded F, the table grade is F. Paper-level grades are computed analogously, averaging across item grades using the same numeric mapping and the same thresholds, while excluding items marked unverifiable, items flagged as judge errors, and items graded F.

Figures are evaluated separately through a vision-capable language model and use the grading rubric in Table 7.

Grade	Near-zero original ($\lvert x\rvert < 0.001$)	Otherwise
A	Absolute difference < 0.002; also assigned when both values are exactly zero	Percentage difference $< 2\%$
B	Absolute difference < 0.02	Percentage difference $< 20\%$
C	Absolute difference < 0.05	Percentage difference $< 40\%$
D	Absolute difference < 0.1	Percentage difference $< 60\%$
E	Absolute difference ≥ 0.1, or signs differ	Percentage difference $\geq 60\%$, or signs differ
F	Assigned if either value is missing	

Table 6: Cell-level grading rules

Grade	Criterion
A	Visually indistinguishable
B	Patterns match, visible differences
C	Recognizable, noticeable gaps
D	Substantially different
E	Fundamentally different
F	Missing, blank, or not verifiable

Table 7: Figure grading rubric

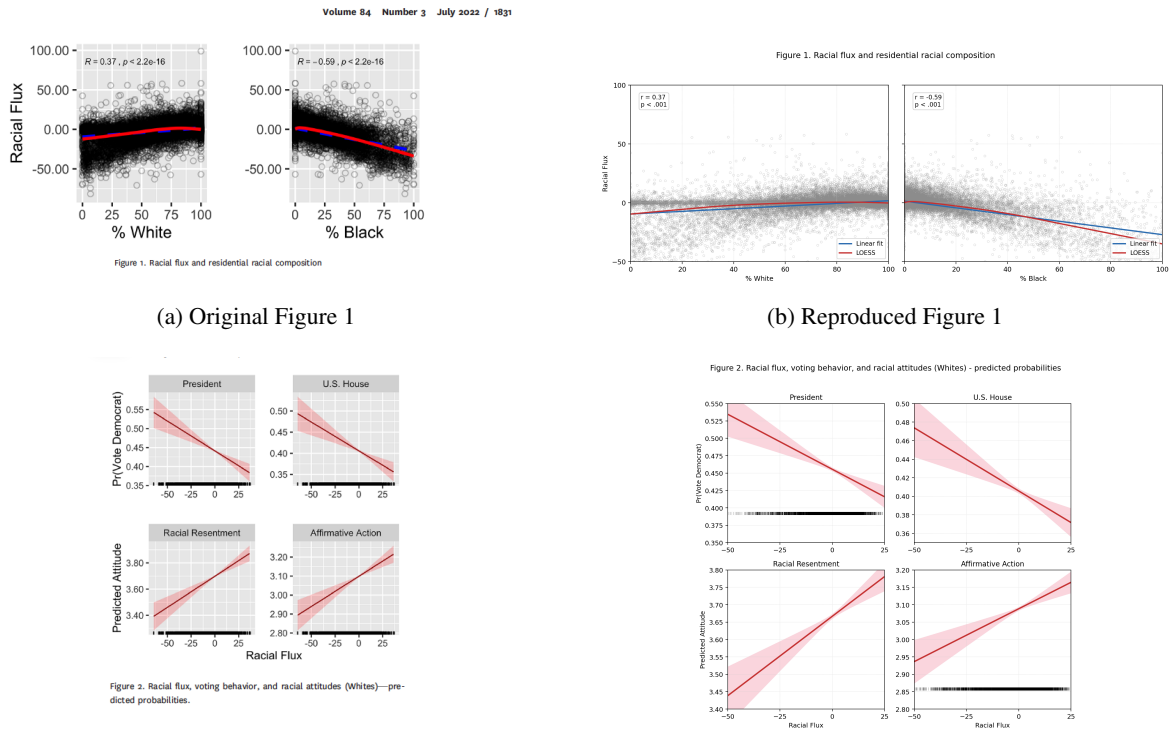
D Additional cell-, table-, and paper-level figures

E Variation in agentic effort: full breakdown

For most models, the main tool call is execution, followed by a mix of reading and writing (Figure 13a). However, OpenCode GPT-5.4 makes more than twice as many tool calls as other setups, spending more calls on execution and many more calls on reading files. When weighting tool calls by volume of text (Figure 13b), the picture looks very different. Most tokens are spent on writing, with almost all of the remaining tokens spent on execution. As a share of tokens, reading, searching, navigation, and other are negligible.

F Multi-run stability

Based on 20 random papers re-run two times (three runs in total) for Claude Code and Codex GPT-5.4.



(a) Original Figure 1

(b) Reproduced Figure 1

(c) Original Figure 2

(d) Reproduced Figure 2

Figure 5: **Agentic reproduction of figure results.** Comparison between original and reproduced figures from the paper "Black Workers in White Places: Daytime Racial Diversity and White Public Opinion." The reproduction was conducted by OpenCode GPT-5.4.

G Leakage analysis

H Correlates of reproduction performance

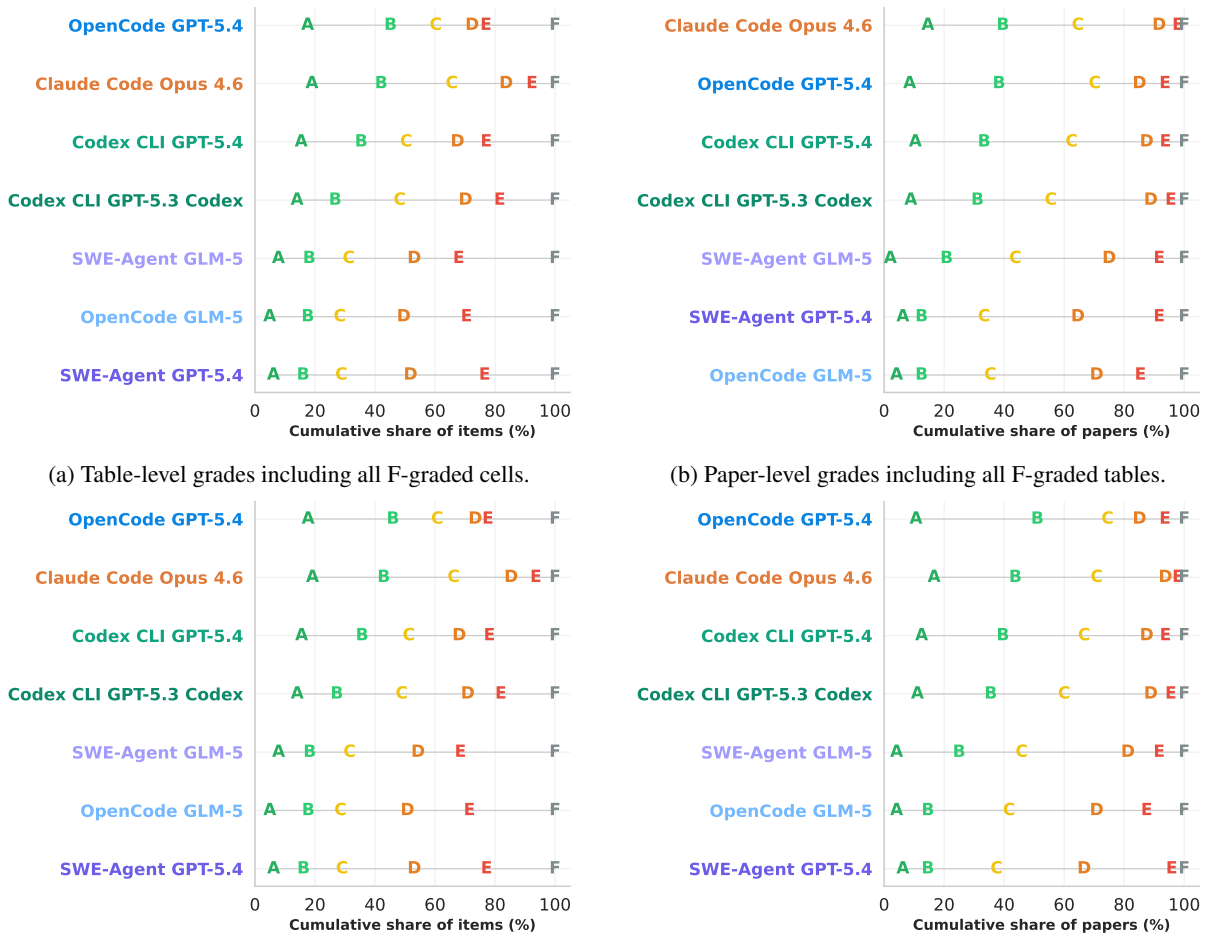
We analyze potential sources of performance variation across papers and agents. Figure 17 shows how frequently two reproducing agents assign the same grade (left) or grades within one step (right). Agents—particularly those developed by the same organization—tend to agree more than would be expected by chance, though between-agent performance varies substantially.

Figure 18 displays the within-paper performance range, which can also be interpreted as a measure of reproduction difficulty. The range varies across papers but remains within two grade steps for the majority, highlighting that some papers are inherently harder to reproduce than others.

Performance also varies across disciplines (Figure 19a): economics papers are generally reproduced less accurately than political science papers. However, since our sample is not random—having been selected by I4Replication based on study reproducibility—this pattern should not be interpreted as a disciplinary characteristic. A similar caveat applies to the comparison between performance and the original authors' code language (Figure 19b).

Finally, we examine whether overall dataset size or the volume of generated code are associated with reproduction performance (Figure 20). No systematic relationship is apparent between dataset size and grade, while a modest positive association is observed between the length of reproduced code and grade.

I Error source analysis



(a) Table-level grades including all F-graded cells.

(b) Paper-level grades including all F-graded tables.

(c) Table-level grades including F-grades for cells with at least one agent with a non-F result.

(d) Paper-level grades including F-grades for tables with at least one agent with a non-F result.

Figure 6: **Table- and paper-level grades including F-grades.** *Aggregated performance naturally shrinks when including empty/not-reproduced tables/items. Notably the best-performing model/scaffold also switches at the paper level.*

	10.1093/ej/ueab069 — Table 3a		10.1257/pol.20200559 — Table 2		10.1093/ej/ueab102 — Table 3	
	Behavior	**Gr.**	**Behavior**	**Gr.**	**Behavior**	**Gr.**
Claude code behavior	Agent assumes id for democrats is party=100	**C**	Agent assumes that codes A1 and A2 indicate democrat support	**D**	Uses `linearmodels'` `diagnostics.f.stat`	**D**
OpenCode 5.4 behavior	Agent assumes id for democrats is party=200	**B**	Agent assumes that codes A4 and A5 indicate democrat support	**B**	Manually reconstructs the (Kleibergen-Paap) Wald F-statistic	**B**
Correct behavior	Democrats id is party=200		A4 and A5 indicates democrat support		Stata uses `ivreg2` (Kleibergen-Paap) `widstat` F-statistic	
Error Attribution	Paper underspecifies code by not disclosing how party is coded in data		Paper underspecifies code by not disclosing how party is coded in data		Paper underspecifies F-statistic type	

Table 8: **Examples of discrepancies of paper underspecification.** *The examples illustrate how agent behavior can vary with an underspecified task.*

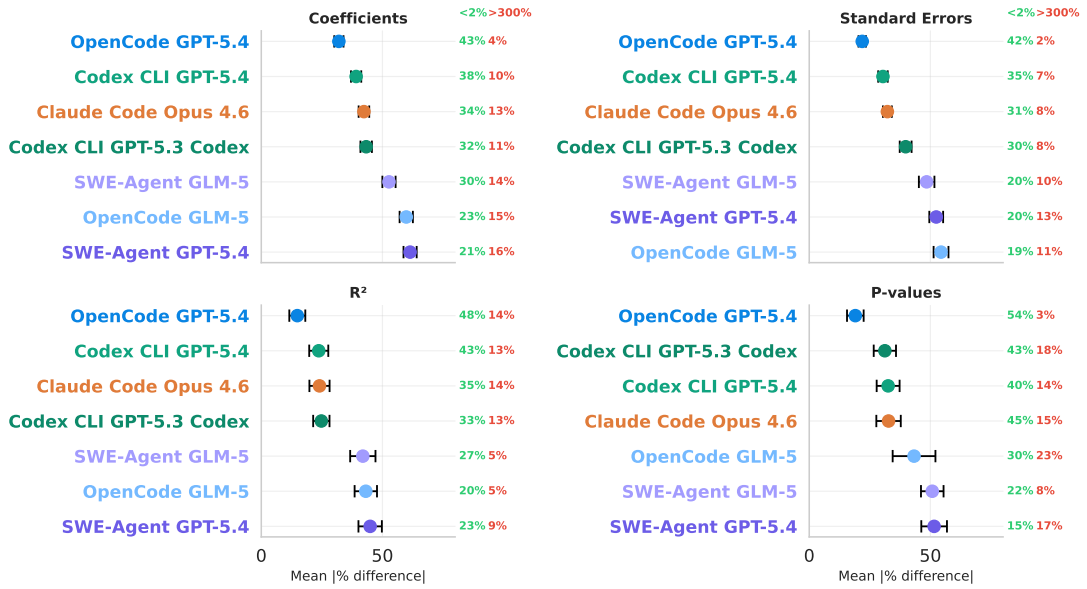


Figure 7: **Mean percentage differences between original and reproduced, by statistic type.** *The strongest models can reproduce more than 40% of all coefficients and standard errors exactly. Note that these statistics are only produced when the original table contains them, and therefore the number of observations per table varies. We drop observations beyond a 300% difference, as these outliers frequently are due to scaling issues. Green and red numbers next to the plot indicate both the share of these large outliers (red) and the share that are (close to) perfectly reproduced (green).*

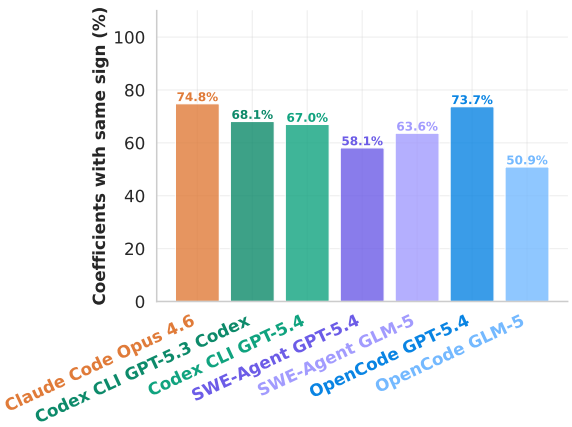


Figure 8: **Share with correct sign, including missing values.** *Share of reproduced coefficients with the same sign as the original paper, including coefficients that are not replicated in the denominator.*

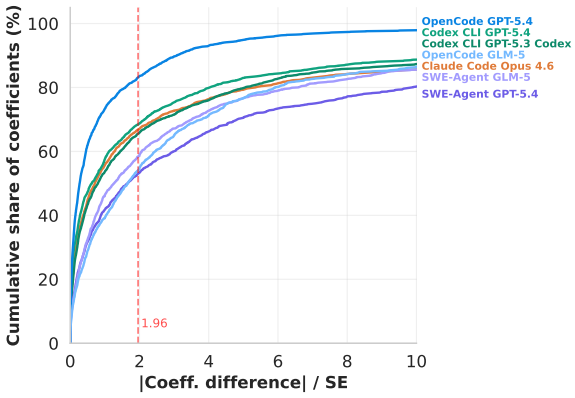


Figure 9: **CDF of SE-adjusted coefficient differences, excluding descriptive-statistics tables.** *Cumulative distribution of |coefficient difference| / SE across approaches when removing descriptive tables. The dashed red line marks 1.96 (95% CI threshold). Neither the magnitude nor the order of results changes meaningfully.*

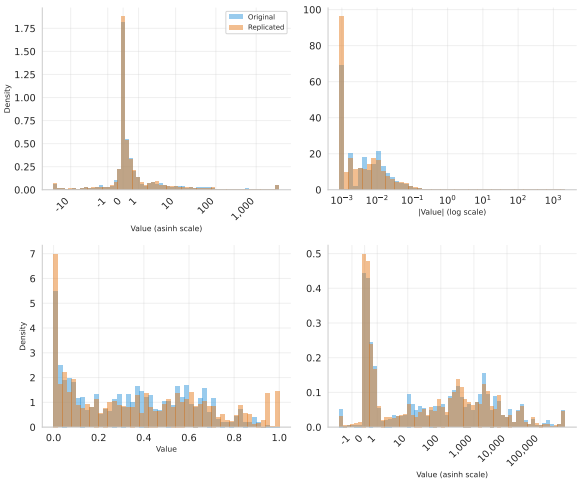


Figure 10: **Histograms of original and reproduced statistics.** *Across various transformations of the statistics, the distributions of the original and reproduced statistics are quite similar.*

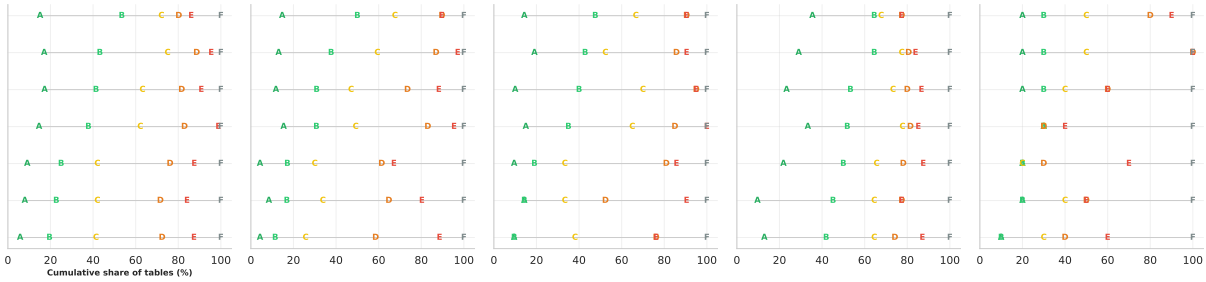
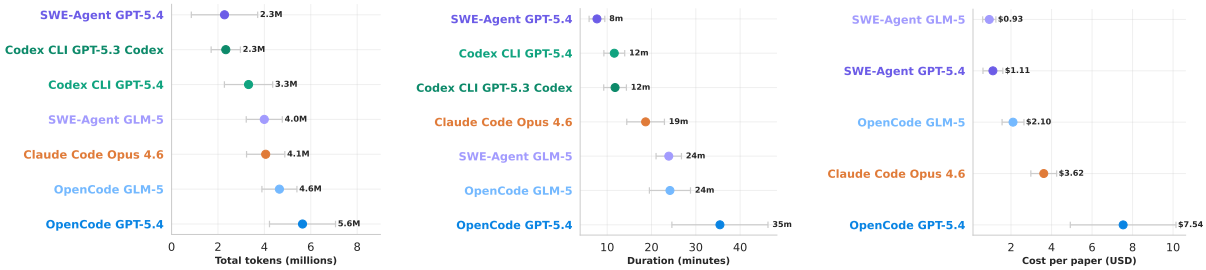


Figure 11: **Grade distributions by table type (including all F-grades).**

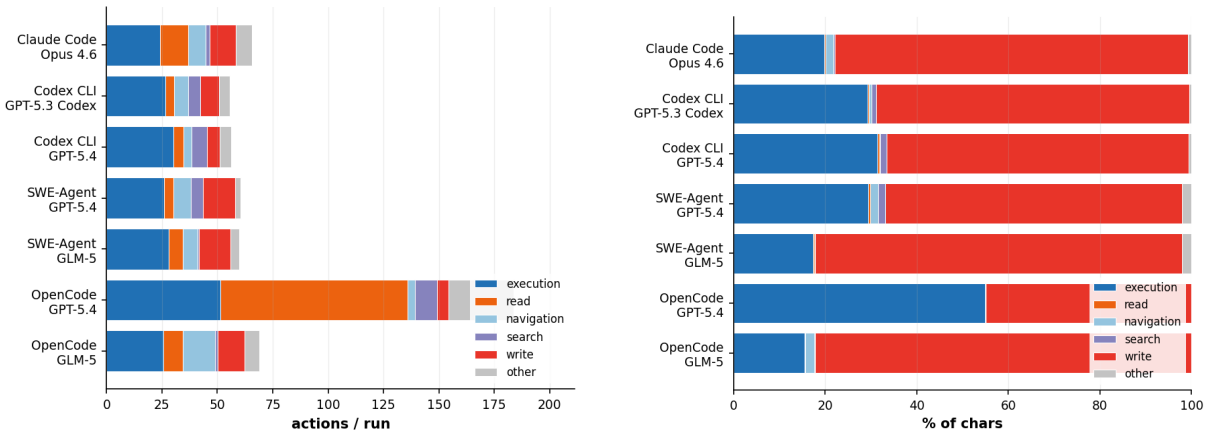


(a) Token usage by run

(b) Duration of runs

(c) Costs of runs

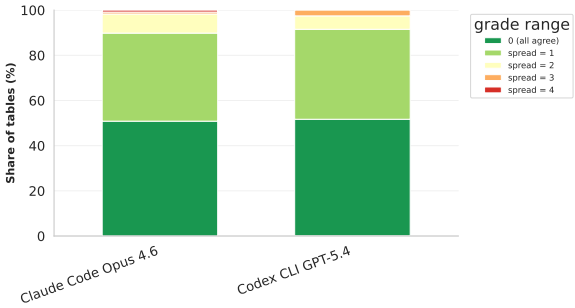
Figure 12: **Token usage, run durations, and run costs, by agent.** *Error bars indicate 95% confidence intervals. Token usage includes cached and uncached input and output tokens. Codex CLI (through subscription) did not report usage costs. OpenCode in combination with GPT 5.4 consumes more tokens, takes longer, and costs more.*



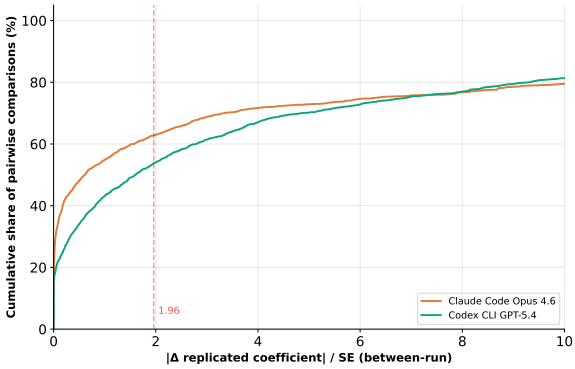
(a) Number of tool-call actions per run

(b) Relative volume of text emitted

Figure 13: **Effort allocation across tasks.** *The figures show the number of tool-call actions taken by the agents (left) and the relative volume of text (number of characters) emitted (right), colored by function category. Most tokens are spent on writing and execution; reading, searching, navigation, and other tasks are negligible by token volume. While GPT-5.4 mostly favors inline Python execution via heredocs, GLM-5 writes scripts to disk before execution, explaining the larger fraction of characters spent on writing files.*

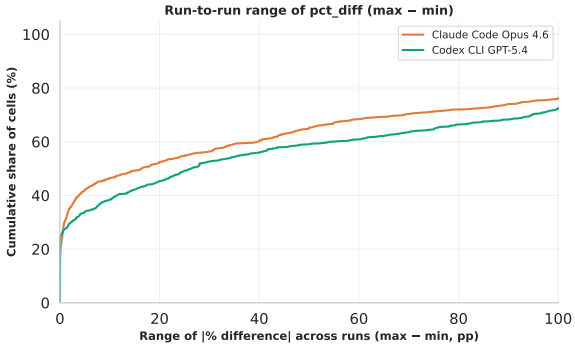


(a) Distribution of table-level grade differences

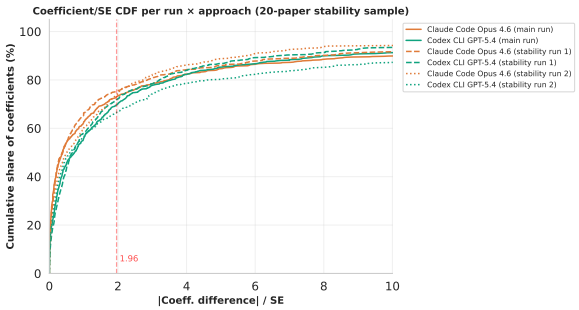


(b) Cumulative distribution of absolute coefficient differences across independent reproduction runs scaled by original standard error.

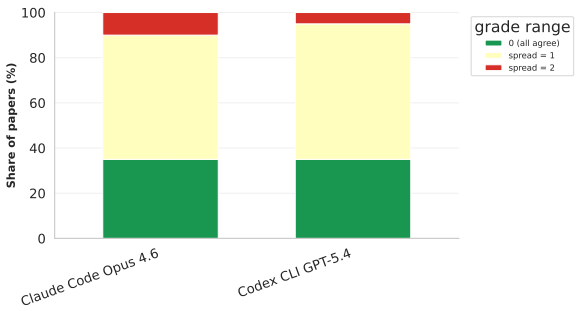
Figure 14: **Stability of results between multiple reproduction runs.** *Both agents reproduce 20 random papers two additional times to a total of three runs. The grade range is computed as the difference between highest and lowest grade between runs excluding F grades. Panel B shows, for each coefficient cell that an agent replicated in two or more runs, the pairwise absolute difference between the replicated values normalized by the original paper's standard error. The CDF reports the share of pairwise comparisons at or below each threshold.*



(a) For y% of cells the difference between the best and the worst reproduction attempt has been at most x%.



(b) Distribution of t-statistics between runs.



(c) Distribution of paper-grade differences.

Figure 15: **Additional statistics on stability between runs.**

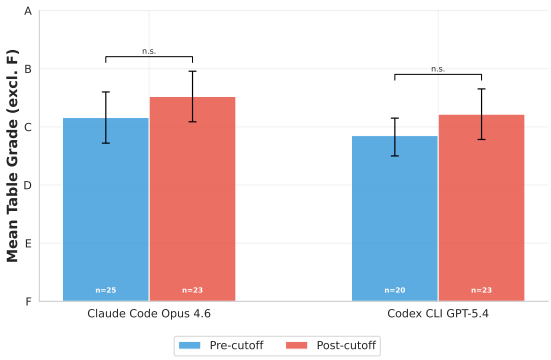


Figure 16: **Pre-training leakage evaluation.** *Average table grades for samples of papers published before and after the model knowledge cutoff. No statistical difference suggests that performance of models in main analysis is not driven by pre-training leakage.*

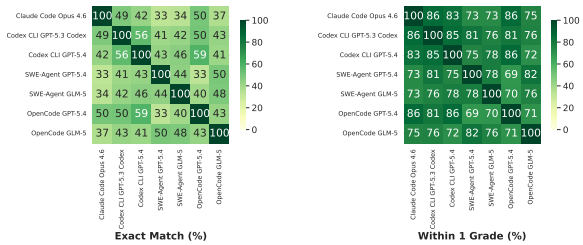


Figure 17: **Inter-agent agreement at the paper level.** *Left: share of papers for which two agents assign the same grade. Right: share of papers for which grades are within one step.*

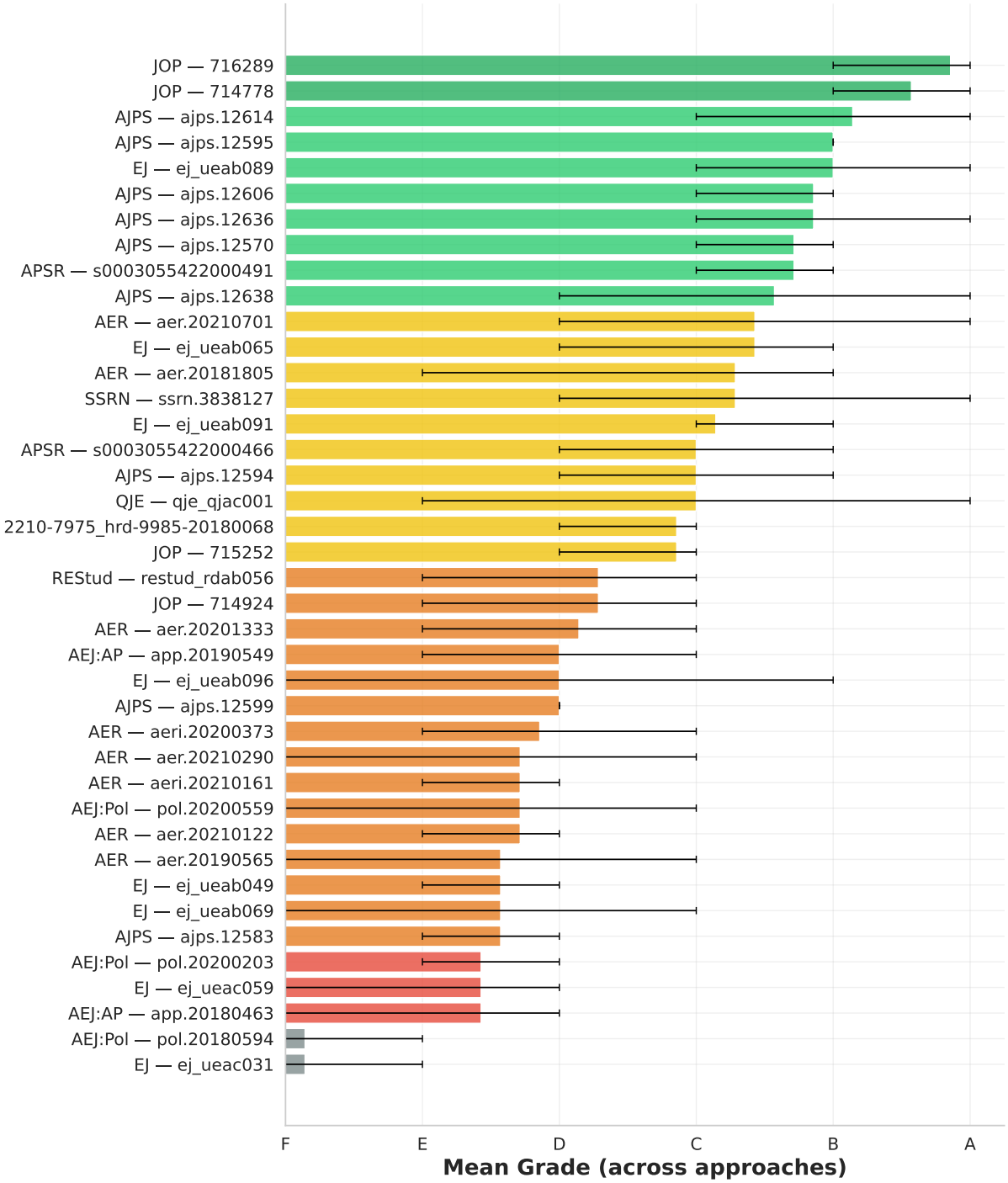
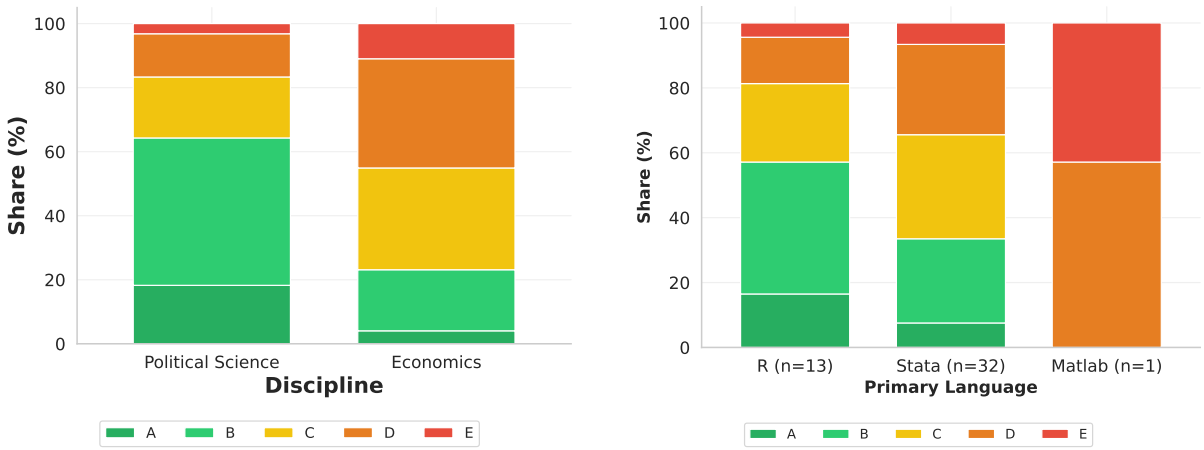


Figure 18: **Within-paper performance range across all agent–model combinations,** *sorted by mean grade. Each bar spans the worst to best grade assigned by any agent.*



(a) Grade by discipline

(b) Grade by primary language

Figure 19: **Paper-level grades by social science discipline and programming language used in the original reproduction package.** *Only papers with a single programming language considered.*

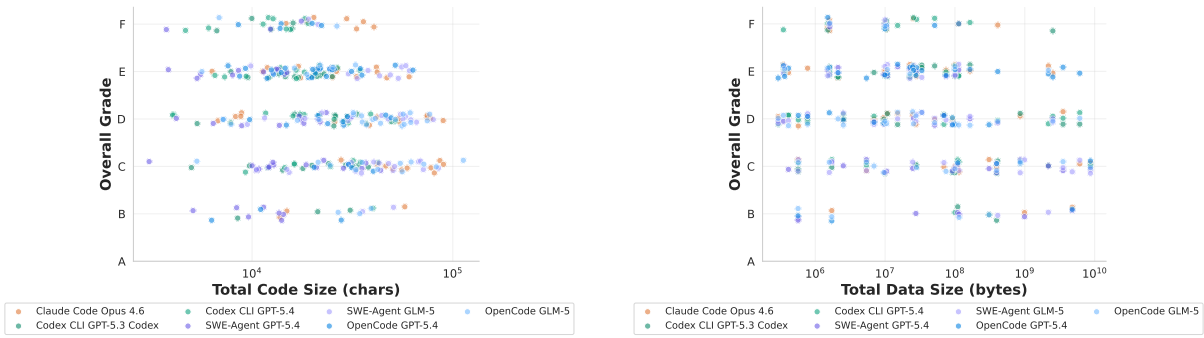


Figure 20: **Paper-level grade by total code characters produced by the reproducing agent and the size of all datasets required for reproduction.**

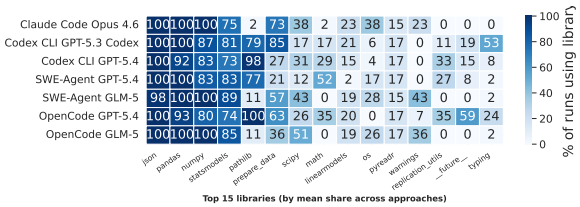


Figure 21: **Incidence of Python imports across runs by agent.** *Note that* `prepare_data` *and* `replication_utils` *are Python scripts that the agent created.*

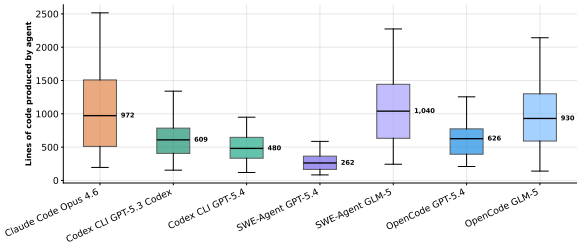


Figure 22: **Distribution of lines of code produced by agents.**

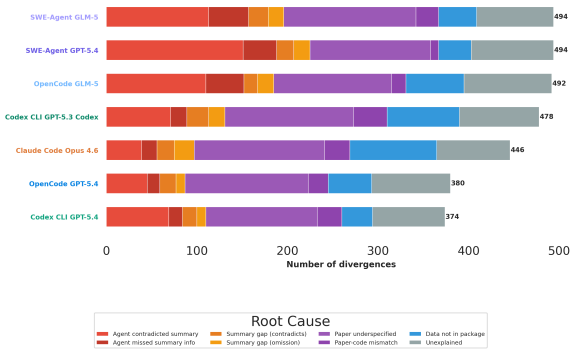


Figure 23: Comparison of error source attribution pipeline results between agents.

Table 9: Example for output of error source pipeline identifying divergences, assigning severity, a stage of failure and a type of error for paper 10.1086_715252

Output	**Sev.**	**Fail stage**	**Fail type**	*Agent*	**Description**
Table 1	critical	Sum. ≠ Agent	✗	*Claude*	Agent uses simple AMCE-difference approach instead of the parallel design causal mediation estimator (med.para) with covariates, and reports two-sided instead of one-sided p-values.
Table 1	critical	Code ≠ Paper	○	*Codex 5.3*	The Python replicator substitutes the original parallel-design mediation/randomization-inference procedure with a pooled round-level product-of-coefficients OLS routine, so the indirect-effect estimates and their reported p-values are not computed the way the R code does.
Table 1	critical	Sum. ≠ Agent	✗	*Codex 5.4*	The agent maps the mediator-specific natural and controlled columns to the wrong experiment indicators, so the natural columns use ‘exp_id_*_con‘ and the controlled columns use ‘exp_id_*_nat‘.
Table 1	critical	Code ≠ Paper	○	*OC 5.4*	The Python replicator replaces the original covariate-adjusted ‘med.para.ri()‘ mediation analysis on ‘exp_id_* == 1‘ versus ‘== 2‘ experiment arms with round-based treatment-control mean differences and separate within-round permutation tests.
Table 1	critical	Code ≠ Paper	○	*SWE 5.4*	The Python replicator does not implement the original mediation routine for Table 1 and instead fills every indirect-effect, parenthesized uncertainty, and total-effect cell from a simple punishment mean-difference plus bootstrap computed on hand-picked round subsets.
Table 1	medium	Code ≠ Paper	○	*OC GLM*	The Python code reports HC1 standard errors in the inference rows, while the original R code reports randomization-inference tail probabilities (‘indirect.greater‘ and ‘total.greater‘).
Table 1	critical	Code ≠ Paper	○	*SWE GLM*	The Python replicator does not implement the original ‘med.para.ri‘ parallel-design mediation procedure for Table 1; instead it builds columns from ad hoc round filters and estimates indirect effects with two OLS regressions on ‘rate_moral‘/‘rate_competence‘, omitting the original experiment-ID split, covariate set, and permutation-based inference.
Table 1	medium	Code ≠ Paper	○	*Codex 5.3*	For total effects, the Python code estimates a simple treatment coefficient on pooled round subsets, while the R code estimates the total effect only on the treatment-only experiment arm selected by each ‘exp_id_*‘ variable and includes the specified conjoint covariates.
Table 1	critical	Code ≠ Paper	○	*Codex 5.4*	The agent does not implement the paper’s mediation estimator with covariate adjustment and randomization-inference p-values; it instead fits a pooled ‘vote ~ punishment * group‘ regression and reports cluster-robust normal p-values.

continued on next page

(continued)

Output	Sev.	Fail stage	Fail type	*Agent*	Description
Table 1	medium	Code ≠ Paper	○	*OC* *GLM*	The Python replicator replaces the mediation estimator with a simple difference between treatment coefficients from experiment arms, omitting the covariate-adjusted total-effect regression and the mediator-level weighted direct-effect calculation used in the R code.

Notes: Fail type: ✗ contradicts (direct conflict); ○ omission (upstream specifies, downstream silent); ? unclear. Agents: Claude = Claude Code Opus 4.6; Codex 5.3/5.4 = Codex CLI; OC = OpenCode; SWE = SWE-Agent; GLM = GLM-5.

J Trace analysis and guardrails

J.1 Setup

Each agent operates in a workspace containing: `TASK.md` (replication instructions extracted from the paper), `methodology_summary.json` (structured methodology context), `table_templates/*.json` (expected output structure), a symlink to the original dataset, and agent-specific instruction files. These files contain paper-derived content by design. The original replication package and paper PDF reside on the same machine but outside the workspace boundary. Web access was restricted using the agent scaffold's configuration to the extent possible.

J.2 Trace analysis

Runs are analyzed by parsing execution logs and extracting tool calls. The agents differ in how they expose file operations: Claude Code and OpenCode provide structured `Read`/`Write`/`Edit` tools alongside bash, Codex exposes bash together with a programmatic file editor, and SWE-Agent operates primarily through bash.

To normalize across these interfaces, each bash command is split into sub-commands at common boundaries (e.g., `&&` chaining), while preserving heredoc bodies as atomic units. Each sub-command is then classified by its first token into one of:

- `exec`: executing code (e.g., `python`, `Rscript`, `make`)
- `read`: reading files (e.g., `cat`, `head`, `sed`)
- `navigation`: filesystem navigation (e.g., `ls`, `cd`, `pwd`)
- `search`: searching (e.g., `grep`, `find`, `rg`)
- `write`: writing files (e.g., `echo`, `cp`, `tee`, `cat` with >)
- `other`: everything else

For each run we measure two quantities: the *action count* (the number of tool calls or sub-commands), and the *tool-call character count* (the number of characters the LLM generated per tool call), as illustrated in Figure 13.

J.3 Guardrail audit

For the replication runs we extract the event trace (all tool calls with inputs and truncated outputs) and the workspace file tree and contents. Guardrail adherence is assessed via two independent signals:

Regex scan. We extract all absolute file paths and URLs from the raw event trace using regular expressions. Each path is classified as: *allowed_workspace* (inside the workspace), *allowed_data* (the mounted dataset symlink), *forbidden_replication_package* (inside `papers/<id>/replication_package/`), *forbidden_paper_pdf* (the paper file), or *forbidden_external* (any other path outside the workspace). URLs and web-access keywords (`curl`, `wget`, `requests`) are flagged separately. This provides a deterministic baseline independent of LLM judgment.

LLM review. In an effort to flag additional runs for potential violations we prompt GPT-5.4-mini with the normalized event trace and workspace artifacts to identify access breaches. Each breach is classified as:

- *external_result_lookup*: fetching results from the web, APIs, or forbidden files outside the workspace;
- *forbidden_paper_access*: reading the published paper or prior replication results;
- *forbidden_code_access*: reading original replication scripts or code from outside the workspace.

Each breach includes the artifact and line range, severity (low/medium/high), confidence (low/medium/high), and an evidence chain. The overall run assessment ranges from *clean* to *severe_violation*, with *insufficient_evidence* for runs where logs are missing or truncated. The model is explicitly instructed that all workspace files are allowed inputs and that incorrect replications are not breaches. In practice we find the LLM-based review to be over-inclusive; severe suspected cases are manually inspected and rerun if breaches were found.

J.4 Hardcoding audit

For each run, we provide GPT-5.4-mini with all generated Python scripts from the workspace alongside the methodology summary and table templates

for context. The model identifies numeric literals that appear in the script's output as statistical results but have no computation path from the dataset. Each run is classified from *clean* to *severe_hardcoding*, with *insufficient_evidence* when Python files are missing or incomplete. In Figure 24 we classify the runs per agent scaffolding and model into their respective hardcoding categories. To assess whether hardcoding inflates replication grades, we compare runs classified as clean ($n = 299$) against all non-clean runs ($n = 37$). Clean runs achieve a mean grade of 3.02 ($\approx$ C) compared to 2.50 ($\approx$ D) for non-clean runs. Runs flagged for hardcoding perform no better—and if anything worse—than clean runs, providing no evidence that hardcoding inflates replication outcomes, though we acknowledge that hardcoding remains an undesirable behavior.

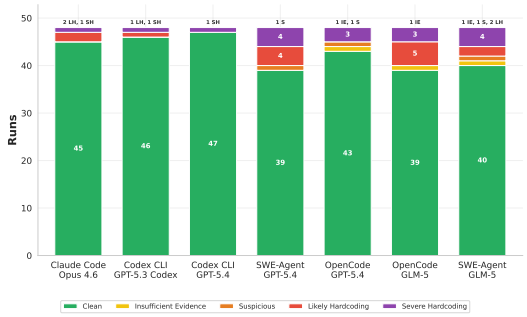


Figure 24: **Hardcoding assessment outcomes by model and agent scaffold.** *Each stacked bar shows the number of runs classified from* clean *to* severe hardcoding.

K Code snippets from reproduction agents

L LLM usage

The conception and argumentation for this paper came entirely from the authors. All sections in the main text were originally drafted by humans, with occasional light revision for grammar and length by LLMs in some cases. Most figures and some tables were created with LLM assistance; the guardrails appendix also includes some AI-generated descriptions of code. On the coding side, AI agents were used to assist in the creation of the pipeline and the analysis.

```python
def main():
    df = main_sample(load_data())
    table = load_template(2)
    formulas = [
        "OUTCOME ~ tbula_3rd + tcoh + treat",
        "OUTCOME ~ treat + C(year_3rd) + C(bula_3rd)",
        "OUTCOME ~ treat + C(year_3rd) + C(bula_3rd) + C(cityno)",
    ]

    for col_idx, formula in enumerate(formulas):
        for outcome, row_coef, row_se in OUTCOMES:
            used = df[[outcome, "tbula_3rd", "tcoh", "treat", "year_3rd", "bula_3rd", "cityno"]].dropna().copy()
            result = run_ols(formula.replace("OUTCOME", outcome), used)
            fill_coef_se(
                table,
                row_coef,
                row_se,
                col_idx,
                result.params["treat"],
                result.bse["treat"],
                result.pvalues["treat"],
            )
        set_cell(table, 20, col_idx, format_observations(len(used)), len(used))

    write_table(table, Path("table_2.json"))
```

(a) Codex 5.4 — *table_2.py*

```python
def run_ols(formula, data, cluster_col="cityno", weights=None):
    data = data.dropna().copy()
    if weights is None:
        res = smf.ols(formula, data=data).fit(
            cov_type="cluster",
            cov_kwds={"groups": data[cluster_col]},
        )
    else:
        res = smf.wls(formula, data=data, weights=data[weights]).fit(
            cov_type="cluster",
            cov_kwds={"groups": data[cluster_col]},
        )
    return res
```

(b) Codex 5.4 — *replication_utils.py*

```python
def add_entropy_weights(aggregates: pd.DataFrame, features: list[str]) -> pd.DataFrame:
    pre_period = (
        aggregates.loc[aggregates["year_3rd"].isin([2006, 2007])]
        .groupby("municipality_id")[features]
        .mean()
    )
    municipality_state = aggregates[["municipality_id", "state_3rd"]].drop_duplicates().set_index("municipality_id")
    treated_ids = municipality_state.index[municipality_state["state_3rd"].eq(13)]
    control_ids = municipality_state.index[~municipality_state["state_3rd"].eq(13)]

    target_means = pre_period.loc[treated_ids, features].mean(axis=0)
    control_weights = entropy_balance(pre_period.loc[control_ids, features], target_means)

    weights = pd.Series(1.0, index=pre_period.index, name="ebal_weight")
    weights.loc[control_ids] = control_weights
    return aggregates.merge(weights, left_on="municipality_id", right_index=True, how="left")


def main() -> None:
    df = load_data()
    aggregates = table7_aggregates(df)
    table = load_json(TEMPLATE_PATH)

    for col_index, column_label in enumerate(table["column_labels"]):
        weighted = add_entropy_weights(aggregates, VERSION_FEATURES[column_label])
        for outcome, coef_row, se_row in OUTCOME_ROWS:
            result = fit_ols(
                weighted,
                f"{outcome} ~ voucher + C(year_3rd) + C(municipality_id)",
                weight_col="ebal_weight",
            )
            bundle = coef_bundle(result, "voucher", add_stars=False)
            set_cell(
                table,
                coef_row,
                col_index,
                raw_text=bundle["coef_text"],
                numeric_value=bundle["coef"],
            )
            set_cell(
                table,
                se_row,
                col_index,
                raw_text=bundle["se_text"],
                numeric_value=bundle["se"],
            )
```

(c) OpenCode GPT 5.4 — *table_7.py*

```python
def entropy_balance(
    control_features: pd.DataFrame,
    treated_target: pd.Series,
    *,
    maxiter: int = 10000,
    tol: float = 1e-10,
) -> np.ndarray:
    x = np.asarray(control_features, dtype=float)
    target = np.asarray(treated_target, dtype=float)

    def moment_gap(lmbda: np.ndarray) -> np.ndarray:
        scores = x @ lmbda
        scores -= scores.max()
        weights = np.exp(scores)
        weights /= weights.sum()
        return weights @ x - target

    sol = optimize.root(moment_gap, np.zeros(x.shape[1]), method="hybr", tol=tol)
    if not sol.success:
        raise RuntimeError(f"Entropy balancing did not converge: {sol.message}")
    scores = x @ sol.x
    scores -= scores.max()
    weights = np.exp(scores)
    weights /= weights.sum()
    weights *= len(weights)
    return weights
```

(d) OpenCode GPT 5.4 — *replication_utils.py*

Figure 25: **Example screenshots from agents reproducing standard econometric methods.** *All screenshots are from reproduction attempts of the paper "The Long-Run Effects of Sports Club Vouchers for Primary School Children" (Marcus, Siedler & Ziebarth—AEJ:Pol, 2022). Panel (a) shows the main regression using a TWFE model for DiD analysis. Helper functions like run_ols, format_observation, and write_table are created in a separate file. Panel (b) shows such a helper. Panel (c) shows an alternative empirical approach via an entropy-balanced synthetic control, applying a shared function shown in Panel (d).*

M Prompts

C1. Extraction Prompt

```
You are a methodological extraction specialist for social science research papers.

Your task: extract the methodology, structure, and specifications from academic papers so that
a replicator who cannot see the paper can reproduce every table and figure. Extract NO results.

## 1. What to extract

- Research questions
- Data description, source, sample size, time period
- Data processing steps: BOTH general steps (applied to all analyses) AND per-table/per-figure
  steps (sample restrictions, variable construction, or filtering specific to one table or figure)
- Per-table: exact column headers, exact row labels, panel structure, caption, notes,
  regression specifications, data source (if different tables use different datasets)
- Per-figure: full caption, plot type, axis labels, legend/series entries, subplot structure,
  visual details (approximate axis ranges, reference lines, line styles, color conventions)
- Per-regression-spec: model type, dependent variable, independent variables, controls,
  fixed effects, clustering, sample restrictions, equation, variable definitions

Extract ALL tables and ALL data-based figures. Skip only purely conceptual visualizations
(flow diagrams, frameworks, screenshots, photos, maps). Do not skip tables because they seem
simple - summary statistics, balance tables, and cross-tabulations must all be extracted.

## 2. What NOT to extract (results)

Never include: regression coefficients, standard errors, t-statistics, p-values, significance
stars, point estimates, confidence intervals, effect sizes, or descriptions of direction/magnitude.

DO include: table/figure structure (headers, labels, panels), design parameters (sample sizes,
thresholds, variable names), and descriptive labels ("N", "R-squared", "Controls", "Yes"/"No").

## 3. Data processing and cleaning steps

This is critical for replication. Extract every step needed to go from the raw data files to
each analysis-ready sample. Separate these into:

**General steps** (applied before any specific analysis):
- File loading, merging, and reshaping (which files, which keys)
- Variable construction (new columns derived from raw data)
- Data cleaning: missing value treatment, outlier removal, winsorization
- Sample filtering: time period restrictions, geographic restrictions, demographic restrictions
- Transformations: log transforms, standardization, encoding

**Per-table / per-figure steps** (use `sample_restrictions` on each table/figure spec):
- Additional filtering specific to one table (e.g., "Table 3 restricts to males aged 25-64")
- Subsample definitions for different columns (e.g., "Columns 1-3 use the full sample,
  columns 4-6 restrict to the treatment group")
- Variable construction specific to one analysis (e.g., "For Figure 2, compute rolling 30-day
  averages of daily returns")

Use the actual variable names from the dataset where possible.

## 4. Regression specifications

Attach each regression spec to the table or figure that displays its results. Include one spec
per distinct model variant (e.g., different controls or subsamples across columns).

For each spec:
- Use actual variable names from the paper, not generic descriptions.
  Write `dependent_var: "log(wage)"`, not `dependent_var: "outcome variable"`.
- `controls`: enumerate every control variable individually. Never write aggregate descriptions
  like "socio-demographic controls"--- list each variable: ["age", "female", "education_level", ...].
- `equation_latex`: copy the exact equation from the paper in LaTeX notation, preserving all
  subscripts, Greek letters, and notation. If no explicit equation is given, write one from the
  prose description. Example:
  `Y_i = \\beta_0 + \\beta_1 X_i + \\gamma Z_i + \\delta_j + \\epsilon_i`
- `variable_definitions`: define every symbol verbally, separated by semicolons. Include all
  information the paper provides: definition, construction, coding, unit, sign convention,
  omitted categories, data source. Only include what is explicitly stated--- do not infer.
  Example: `A_i^T: acceptance of targeted tax (1 if not "No", as defined in Section 2.3);
  c_i: income-threshold fixed effects (4 levels: bottom 20/30/40/50 percentile)`
- `sample_restrictions`: extract the exact condition including any mathematical formulation
  and expected sample size.
  BAD: "Among invalidated respondents"
  GOOD: "Among invalidated respondents, defined as sgn(g_i) != sgn(gamma_hat_i) (Section 4.1). N = 1,365."

## 5. Cross-reference resolution

When anything references content elsewhere ("see Appendix H", "controls listed in Table A3",
"as defined in Section 2"), go to that location and extract the actual content. Never leave
unresolved references--- the replicator cannot see the paper. This is especially important for
control variable lists: find the appendix and enumerate every variable individually.
```

```
## 6. Variable completeness

Extract ALL variables: dependent, independent, controls, instruments, weights, and constructed
variables. For each, combine information from all locations in the paper (methodology section,
table notes, appendices). The goal: a replicator should be able to construct every variable
exactly as the authors did based solely on your extraction.

## 7. Table and figure precision

- Copy EXACT column headers and row labels as printed. Count columns and rows precisely.
- Include panel structure (Panel A / Panel B) when present.
- Copy full captions and table notes (excluding notes about specific coefficient values).
- For figures: note approximate axis ranges, reference lines, and line style conventions.
- If different tables use different datasets, specify the data source per table.""


EXTRACTION_USER_PROMPT = """Extract the methodology from this paper. Follow the system instructions.

## Detected Table Captions:
{table_captions}

## Detected Figure Captions:
{figure_captions}

## Paper Text:
{paper_text}"""


TEMPLATE_GENERATION_SYSTEM_PROMPT = """You are a structural template specialist for academic paper tables and figures.

Generate templates that faithfully reproduce the exact layout of tables and figures from a paper.

## Table templates

- EXACTLY the same number of columns and rows as the original table.
- Use the EXACT column headers and row labels from the paper.
- Cell content rules (check the ORIGINAL TABLE in the paper for each cell):
  - **XXX**: computed result (coefficient, statistic, count, mean, etc.)
  - **(XXX)**: standard error in parentheses
  - **Empty**: leave blank if blank in the original (do NOT use --- or placeholders)
  - **Literal text**: copy "Yes", "No", checkmarks, or labels as-is
- Include panel headers (Panel A, Panel B) as spanning rows when present.
- Include the full caption above the table.

## Figure templates

- Matplotlib code skeleton that makes the intended visual style unambiguous.
- Use the correct plot function: `ax.plot()`, `ax.bar()`, `ax.hist()`, `ax.scatter()`, etc.
- Set line styles, markers, colors for each series (use `tab:blue`, `tab:orange`, etc.).
- Set approximate axis ranges from the paper (`ax.set_xlim()`, `ax.set_ylim()`).
- Add reference lines (axvline, axhline) if the paper has them.
- Use actual series names / legend labels from the paper.
- Include the full caption as the title.
- Set up subplots correctly if the figure has panels.
- NO actual data arrays--- use comments like `# TODO: fill with data`.
- The skeleton should be runnable (producing an empty styled plot) even without data."""


TEMPLATE_GENERATION_USER_PROMPT = """Generate structural templates for each table and figure.
Respond with valid JSON.

Go back to the ORIGINAL TABLE in the paper text and reproduce its structure EXACTLY.

Example regression table template (note how Variable C only appears in columns (3)-(4)):

| | (1) | (2) | (3) | (4) |
|---|---|---|---|---|
| Variable A | XXX | XXX | XXX | XXX |
| | (XXX) | (XXX) | (XXX) | (XXX) |
| Variable B | XXX | XXX | XXX | XXX |
| | (XXX) | (XXX) | (XXX) | (XXX) |
| Variable C | | | XXX | XXX |
| | | | (XXX) | (XXX) |
| Controls | No | Yes | No | Yes |
| Observations | XXX | XXX | XXX | XXX |

Example figure skeleton:

```python
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 1, figsize=(10, 6))
# x_obj, density_obj = ... # TODO: compute KDE from data
# ax.plot(x_obj, density_obj, color="tab:blue", linestyle="-", linewidth=2, label="Objective")
ax.set_xlabel("Net gain (EUR/year)", fontsize=12)
ax.set_ylabel("Density", fontsize=12)
ax.set_xlim(-1500, 1500)
ax.axvline(x=0, color="black", linestyle=":", linewidth=0.8)
```
```

```
ax.legend()
plt.tight_layout()
plt.savefig("figure_X.png", dpi=150)
```

Paper Text (excerpt):
{paper_text}

Extracted Tables:
{table_specs_json}

Extracted Figures:
{figure_specs_json}"""

Non-replicable figure types (code-level safety net)

NON_REPLICABLE_KEYWORDS = [
    "flow diagram", "flow chart", "flowchart", "conceptual framework",
    "conceptual diagram", "diagram", "schematic", "screenshot", "photo",
    "photograph", "timeline",
]

C2. Reproduction Prompt

```
# Replication Task

You are given a methodological summary of a research paper and its associated
dataset. Your goal is to replicate the paper's empirical results using only
the methodology description below and the data. You do not have access to the
original paper, its results, or any replication code.

## Data

The dataset is located at: `{data_filename}`

## Methodological Summary

**Paper**: {title} (ID: {paper_id})

**Research Questions**:
{research_questions}

**Data Description**: {data_description}

**Data Context**: {data_context}

{data_source}{time_period}

**Data Processing Steps**:
{processing_steps}

{items_section}

## Constraints

You are in an isolated workspace for fair benchmarking.

1. **Workspace only.** Only read and write files inside this workspace directory.
   Do not access files outside of it or navigate to parent directories.
   **Exception:** the `data/` directory may be a symlink pointing outside the
   workspace -- this is intentional and you should use it freely as your dataset.

2. **No searching for the paper.** Do not search the internet for this paper,
   its authors, its published results, or any replication code or packages.
   Do not search for prior replication attempts.

3. **No searching for results.** Do not look up expected coefficients, effect
   sizes, tables, or figures from this paper. Your replication must be derived
   entirely from the methodology summary and the data provided.

4. **Allowed web use.** You may search for Python library documentation
   (e.g. statsmodels, pandas, matplotlib) and general statistical methods.

5. **Work independently.** Base your replication only on the methodology
   description in this file and the dataset.

## Instructions

1. **Quick data check**: Inspect the data files to confirm column names and
   basic structure. The methodology summary above describes the variables in
   detail, so a brief check should suffice.

2. **Write `prepare_data.py`**: Load and clean the data following the processing
```

```
   steps described above. All scripts can import from this module.

3. **Write and execute one script at a time**: For each item:
   a. Write the script (see output filename specified for each item above)
   b. Execute it
   c. Fix any errors immediately
   d. Move on to the next item once the output file is verified

{table_instructions}{figure_instructions}

4. **R packages**: If the paper's methodology requires R-specific packages
   (e.g. for specialized estimators), you can call R from Python using `rpy2`.

5. Execute every script and verify the output file exists.

6. Save all outputs in the current working directory.

**Reasonable assumptions.** Where the methodology description is incomplete or
ambiguous, you are free to make reasonable assumptions based on common practice
in the field. Document your assumptions briefly in comments.

Focus on substance and accuracy. Match the described methodology as closely as
possible, including sample restrictions, variable transformations, and
statistical specifications.
```

Note that, while Prompt M.2 specifies that web use is allowed for researching "Python library documentation" and "general statistical methods," web search was in fact disabled programmatically at the scaffold level.

C3. Explanation - Extraction of Divergences Prompt

```
    You are an expert at tracing numerical replication failures back to their root code discrepancies.

Your working directory contains two subfolders:
  original_code/   -- original {original_language} replication package (code + data directory)
  agent_code/      -- agent-generated Python replicator

## Target failure
Item: {item_id}  |  Grade: {grade}  |  {n_failed} of {n_total} cells failed

{already_attributed_block}Failed cells still to explain (sorted by |% difference|):
{cell_table}

## Your tasks

1. DATA AVAILABILITY --- Check whether the data files required to produce {item_id} exist in
   the data directory inside original_code/. If any required input file is absent from the
   replication package, set data_available = "missing" and describe the missing file(s).
   Otherwise set data_available = "available".

2. AGENT CODE --- In agent_code/*.py, find the code that computes values for the REMAINING
   cells listed above (ignore already-attributed sections).
   If no such code exists, set agent_behavior = "<description of what is missing>".

3. ORIGINAL CODE --- In original_code/{original_file_glob}, find the equivalent {original_language} code.

4. DISCREPANCIES --- For the REMAINING cells only, identify all distinct root causes.
   Most outputs have one, but multi-panel tables with different specifications may have
   several (e.g. columns 1--3 OLS wrong clustering; columns 4--6 IV wrong instrument).
   If all remaining cells are already explained, return an empty divergences array.
   List one entry per distinct root cause --- do NOT split trivial variations.

5. SECTION MAPPING --- For each discrepancy, list which rows/columns/panels it explains
   (e.g. "All columns", "Columns 1--3 (OLS)", "Panel B"). Maps many cells to one cause.
   Also list each specific affected cell from the cell table above as
   {{"item_id": "{item_id}", "row_label": "...", "column_label": "..."}} using the exact
   labels shown. Include all cells you believe this discrepancy explains.

6. ALSO EXPLAINS --- For each discrepancy, list which other failures it also explains.
   Each entry is either:
     - A plain string "<item_id>" if the discrepancy explains the entire other item, OR
     - An object {{"item_id": "<item_id>", "sections": "<which part>"}} if only partial.
   Be conservative: only include items you are confident share the exact same root cause.

## Other unresolved failures
{remaining_block}

## Divergence taxonomy (use exactly one code)
  S1  Wrong model specification      -- incorrect FE, clustering level, or SE type
  S2  Wrong estimator / inference    -- wrong estimator (OLS vs IV) or missing inference step
  S3  Data source substitution       -- proxy used / required dataset absent from package
  S4  Wrong sample restriction       -- filter missing, wrong condition, or wrong subset
  S5  Wrong variable construction    -- outcome/predictor coded differently than reference
  S6  Missing analysis component     -- required step entirely omitted
```

```
  S8  Wrong merge / transform logic  -- wrong join type, key, duplicate handling, or reshape
  S9  Wrong sequencing               -- steps in wrong order, changing results
  S0  Other                          -- does not fit any category above

## Severity
  minor    -- unlikely to materially affect point estimates or conclusions
  medium   -- could shift estimates noticeably; sign/significance probably stable
  critical -- likely changes sign, significance, or core conclusion of a main result

## Output
Write a JSON object with a "divergences" array to {output_path}.
One entry per distinct root cause found in {item_id} (usually 1, occasionally 2--3).
Write ONLY valid JSON --- no markdown fences, no preamble, no comments.
Escape all special characters properly inside string values.

{{
  "divergences": [
    {{
      "id": {divergence_id},   (use sequential integers starting here for additional entries)
      "output": "{item_id}",
      "description": "<one sentence: what is different>",
      "original_behavior": "<what the original {original_language} code does on this specific point>",
      "agent_behavior": "<what Python does, or the ABSENT description>",
      "original_proof": "<short verbatim {original_proof_label}>",
      "agent_proof": "<short verbatim Python snippet, or ABSENT>",
      "original_location": {{"file": "<{original_file_ext}>", "line": "<line or range>"}},
      "agent_location": {{"file": "<filename.py or ABSENT>", "line": "<line or ABSENT>"}},
      "divergence_type": "<S0|S1|S2|S3|S4|S5|S6|S8|S9>",
      "severity": "<minor|medium|critical>",
      "data_available": "<available|missing>",
      "data_available_note": "<one sentence naming specific file(s) checked or found absent>",
      "explains_sections": ["<e.g. 'All columns', 'Columns 1-3 (OLS)', 'Panel B'>"],
      "affected_cells": [
        {{"item_id": "{item_id}", "row_label": "<exact row label from the cell table>", "column_label": "<exact column label>"}}
      ],
      "also_explains": ["<item_id>", {{"item_id": "<item_id>", "sections": "<which part>"}}]
    }}
  ]
}}
```

C4.1. Explanation - Error source Original Paper vs Code

```
You are verifying whether the paper supports the original {original_language} code's behavior
for a set of replication divergences.

Your working directory contains:
  paper.pdf              --- the published paper
  original_code_files/   --- the original {original_language} replication code (for reference only)

For each divergence you are given:
  - `original_behavior`: what the {original_language} code does for this analysis step
  - `original_proof`: the exact {original_language} code snippet implementing this behavior
  - `original_location`: the file and line number in the original code

The original code is already provided --- you do NOT need to re-read the code files.
Read paper.pdf to determine whether it explicitly states, implies, or is silent
about the behavior described in `original_behavior`.  Classify using exactly one
of these four verdicts:

  consistent   = paper and original code explicitly agree on this specific point
  contradicts  = direct contradiction: the paper explicitly says X AND the original
                 code explicitly does Y, where X ≠ Y
  omission     = the original code implements X, but the paper does not mention X at
                 all --- no contradiction, the paper simply doesn't cover this detail

Rules:
- "contradicts" requires both documents to explicitly address the same point with
  different answers.  The paper being silent on an implementation detail is
  NEVER sufficient for "contradicts" --- use "omission" instead.
- "omission" means the original code (the upstream document here) specifies something
  the paper (the downstream document) doesn't address.  Example: code computes
  monthly standardized values and averages them; paper describes the monthly formula
  but says nothing about aggregation → omission.
- "consistent" if both agree; prefer "omission" when one side is explicit and the other is silent.
- Your note must cite specific evidence: variable name, section heading, line
  number, or direct quote.
- You MUST produce exactly one of these three verdicts for every divergence id. Do not abstain or invent new labels.

EFFICIENCY: Do NOT read the entire paper exhaustively.  Use the divergence
description to identify the relevant section, check that section, form a
well-founded hypothesis, cite the evidence, and move on.
```

C4.2. Explanation - Error source Original Paper vs Extractor

```
You are verifying whether the methodology summary accurately represents what
the paper says, for a set of replication divergences.

Your working directory contains:
  paper.pdf                  --- the published paper
  methodology_summary.json   --- the summary passed to the replicator agent

For each divergence you are given:
  - `original_behavior`: what the original code does for this analysis step
  - `original_proof`: the exact code snippet (for context on what to look for)
  - `original_location`: the file and line number in the original code

The original code is already provided --- you do NOT need to read the code files.
Read paper.pdf and methodology_summary.json to determine whether the paper states
this behavior and whether the summary faithfully represents it.  Classify using
exactly one of these four verdicts:

  consistent   = summary faithfully represents the paper on this point, OR both
                 are silent (summary correctly captured the paper's silence)
  contradicts  = direct contradiction: paper explicitly says X AND summary
                 explicitly says Y ≠ X
  omission     = paper explicitly says X, but the summary does not mention X at
                 all --- the summary dropped information the paper provided

Rules:
- "omission" means the paper (upstream) says something the summary (downstream)
  dropped.  This is the most common failure mode for this check.
- "contradicts" requires both documents to explicitly address the same point
  differently --- not just one being silent where the other speaks.
- If neither the paper nor the summary addresses the point, mark "consistent"
  (they agree by silence --- the break is not here).
- The summary is NOT expected to be more detailed than the paper.  If the paper
  states X at a conceptual level (e.g. "cluster at neighborhood level") and the
  summary conveys the same concept (e.g. "use neighborhood cluster identifiers"),
  mark "consistent" --- even if the summary does not reproduce the exact variable
  names or implementation details that appear in the original code but not in the
  paper.  The standard is semantic faithfulness to what the paper says, not
  completeness of implementation detail.
- To distinguish "consistent" from "omission" at the conceptual level, ask:
  "Would a careful agent reading only the summary know that they should implement X
  (the concept the paper describes)?"  If yes → "consistent" (the concept was
  conveyed, even if less specific).  If no, because the summary says nothing about
  X at all → "omission".  Note: "omission" does not require that the agent would
  actively do the wrong thing; it is sufficient that the summary dropped a concept
  the paper stated, leaving the agent without guidance on that point.
- Your note must cite specific evidence from both documents.
- You MUST produce exactly one of these three verdicts for every divergence id. Do not abstain or invent new labels.

EFFICIENCY: Do NOT read documents exhaustively.  Use the divergence description
to locate the relevant section in each document, check it, form a well-founded
hypothesis, cite the evidence, and move on.
```

C4.3. Explanation - Error source Extractor vs Agent

```
You are verifying whether the agent's Python code implements what the
methodology summary instructs, for a set of replication divergences.

Your working directory contains:
  methodology_summary.json   --- the summary passed to the replicator agent
  agent_code/                --- the agent's Python replication code (for reference only)

For each divergence you are given:
  - `agent_behavior`: what the Python code actually does for this analysis step
  - `agent_proof`: the exact Python code snippet implementing this behavior
  - `agent_location`: the file and line number in agent_code/

The Python code is already provided --- you do NOT need to re-read agent_code/.
Read methodology_summary.json to determine whether it explicitly instructs,
implies, or is silent about the behavior described in `agent_behavior`.  Classify
using exactly one of these four verdicts:

  consistent   = agent follows the summary on this point, OR both are silent
                 (agent correctly followed the summary's omission)
  contradicts  = direct contradiction: summary explicitly says X AND agent code
                 explicitly does Y ≠ X
  omission     = summary explicitly instructs X, but the agent does not implement
                 X at all --- agent omitted something the summary described

Rules:
- Each divergence also includes a `description` field explaining what the step
  is about.  Use this to find the relevant section in methodology_summary.json
  when agent_behavior is "ABSENT" or otherwise unclear.
- "omission" covers agent_behavior = "ABSENT": use `description` to identify
  what analysis step was supposed to be implemented, look it up in the summary,
```

```
  and if the summary describes it → omission.  Only mark "consistent" if the
  summary is also silent on that step.
- "contradicts" requires the summary to explicitly say X AND the agent to do
  something explicitly different --- not just the agent being silent.
- IMPORTANT: if the summary does NOT mention this specific detail at all
  (e.g. the summary is silent about weighting, file choice, variable variant,
  clustering method, or any other implementation detail), mark "consistent" ---
  the agent cannot be expected to follow guidance that was not provided.
  The agent making its own reasonable choice on an unspecified detail is NOT
  a contradiction or omission.
- Your note must cite specific evidence from both the summary and the Python code.
- You MUST produce exactly one of these three verdicts for every divergence id. Do not abstain or invent new labels.

EFFICIENCY: Work from the provided code snippets and the summary text.  Do NOT
exhaustively search through all files.  Form a well-founded hypothesis based on
the evidence given, cite it, and move on.
```

C4.4. Explanation - Error source Data Missing

```
You are checking whether the data required by the original {original_language} code is
available to a Python replicator, for a set of divergences.

Your working directory is the replication data directory.  The original
{original_code_description} are in {code_files_path}.

## Step 1 --- identify what the original code loads

For each divergence, read the original code files and find the data file(s)
loaded for that analysis step.

## Step 2 --- distinguish raw inputs from code-constructed intermediates

Before checking whether a file exists, determine whether it is:

  (a) A RAW SOURCE FILE --- brought in from outside (survey data, official
      statistics, downloaded datasets).  These must exist as-is for any
      replicator.

  (b) A CODE-CONSTRUCTED INTERMEDIATE --- a file that is itself created by
      the original code in the same replication package.  A Python
      replicator would construct this from its own upstream steps, not load a
      pre-built file.

You can identify constructed intermediates by searching the original code for
save/export commands that write that filename.

## Step 3 --- check availability

  · If the required file is a RAW SOURCE FILE: check whether it exists in the
    current directory.  If yes → available; if no → missing.

  · If the required file is a CODE-CONSTRUCTED INTERMEDIATE: instead check
    whether the raw source file(s) that feed into its construction are present.
    If those raw sources exist → available (the agent can construct the
    intermediate from them); if the raw sources are also absent → missing.

  available = the required data (raw or constructable from present sources)
              was accessible; the agent had what it needed and chose incorrectly
  missing   = the required raw data is absent; the agent could not have
              implemented this correctly regardless of effort

Every divergence must receive `available` or `missing` --- no other values.
In your note, name the file(s) you checked and state whether each is a raw
source or a constructed intermediate.
\end{lstlisting}
```