NBER WORKING PAPER SERIES

AI AGENTS AND PROMPT ENGINEERING IN ECONOMETRIC CODING

Sebastian Galiani
Federico Ariel López
Raul A. Sosa

Working Paper 35588
http://www.nber.org/papers/w35588

NATIONAL BUREAU OF ECONOMIC RESEARCH
1050 Massachusetts Avenue
Cambridge, MA 02138
August 2026, Revised November 2026

This research received no external financial support. The authors used the Claude Code and Codex coding agents to write and test the benchmark code, to draft task materials, and to draft and edit the manuscript text. The authors designed the study, directed and reviewed the agents' work, and take full responsibility for the content. The views expressed herein are those of the authors and do not necessarily reflect the views of the National Bureau of Economic Research.

NBER working papers are circulated for discussion and comment purposes. They have not been peer-reviewed or been subject to the review by the NBER Board of Directors that accompanies official NBER publications.

© 2026 by Sebastian Galiani, Federico Ariel López, and Raul A. Sosa. All rights reserved. Short sections of text, not to exceed two paragraphs, may be quoted without explicit permission provided that full credit, including © notice, is given to the source.

AI Agents and Prompt Engineering in Econometric Coding
Sebastian Galiani, Federico Ariel López, and Raul A. Sosa
NBER Working Paper No. 35588
August 2026, Revised November 2026
JEL No. C18, C87

ABSTRACT

We study how large language models write the code for statistical and econometric analysis. We assign six major models 40 different tasks in Stata, R, and Python, with different prompts and different degrees of agency. The prompt is either only the statement of the task or a prompt that has a related code example. The least agentic version, the chatbot, just writes the code, and a more agentic version can run and revise its code and decide when to stop. We find that more agency raises success from 84 to 97 percent. Letting the chatbot revise its script achieves most of this gain. The example and the language matter more for the chatbot than for the agent, which also narrows the differences between models. An agent's runs cost 3.4 times as much as chatbot runs, and the cost of an additional correct answer ranges from$0.02 to $1.13, depending on the model. The agent removes almost all loud failures, which occur when the script does not run or does not return every result. On the other hand, it removes only about half of the silent failures, those in which the script runs but returns a wrong value. As a result, most of the failures that remain are silent, so researchers who use AI agents still need to check the code, even when it seems to work.

Sebastian Galiani
Tulane University
and NBER
sebastian_galiani@yahoo.com

Raul A. Sosa
Universidad de San Andrés
Department of Economics
rsosa@udesa.edu.ar

Federico Ariel López
Universidad de San Andrés
federicoarlopez@gmail.com

Introduction

To answer a research question, social scientists frequently choose a design that identifies a causal effect (Angrist and Pischke, 2010; Garg and Fetzer, 2025). They then write the code in Stata (a statistics package), R, or Python to process the data and compute the estimates. Much of this coding is mechanical (once the design is clearly established), so it is usually delegated to a research assistant or, more recently, to large language models (LLMs). What is left is reviewing, correcting, and standardizing. Coding errors are a common problem in empirical work. They are also difficult to detect, and they include semantic bugs, in which the code usually runs but does not compute what was intended (Tan et al., 2014). An audit of 110 studies published in economics and political science journals found that coding errors affect about a quarter of them and can be present even when the posted code reproduces the published numbers exactly (Brodeur et al., 2026*b*). LLMs already seem to raise the productivity of professional software developers (Cui et al., 2026) and can now also write the code of an empirical analysis (Grundl, 2026). But, just like researchers or their research assistants doing the coding, LLMs may introduce such errors into the estimation scripts. If there is little human oversight, AI-led teams reproduce fewer published results and find fewer coding errors than human-only teams (Brodeur et al., 2026*a*).

The appearance and widespread adoption of these AI tools for research open up a few questions. What will be left for us humans to do? Will we become prompt-writers? Will our human input still have value as these systems become more powerful and more *agentic*? And what happens to everything we learned about a particular programming language? In this paper we study the interaction between agency, the prompt, and the language in determining the success and cost of six leading LLMs that write econometric code. In our design, the researcher has already chosen the question, the method to answer it, and the identification assumptions; the language model just receives a task and writes the code that implements what was asked. Our main finding is that a model that runs its own code removes most errors that stop the script or leave a value missing, and that most of the errors that remain give no sign that they are wrong.

Writing code can be thought of as having five stages: 1) translating the request into a plan, 2) writing the script, 3) running it, 4) inspecting the output, and 5) revising the script when something is wrong. A basic chatbot is the least capable version of an AI system. It just answers with a script for the researcher to run. It performs only the first two stages. The more capable systems are more *agentic*. *Agency* is the extent to which the model performs the last three stages itself before submitting a result, and these stages require code execution and a response to its output.

Most individual use of LLMs still takes place through a chat window. ChatGPT alone reaches about a tenth of the world's adults (Chatterji et al., 2025). However, fewer than 1 percent of its active individual users work with the Codex agent (Johnston et al., 2026). The adoption of coding agents is growing quickly (Massenkoff et al., 2026; Johnston et al., 2026). Among surveyed quantitative social scientists, 81 percent have used AI chatbots in their research and 20 percent use coding agents regularly (Lyttelton, Massenkoff and Wilmers, 2026).

In our study we have three *agency* conditions, each adding progressively more capability (Figure 1). The first, least capable one, is the *chatbot* that writes one script and finishes. The next one is *self-repair*, where a model gets more chances to revise its script, while it neither runs the script nor decides when to stop itself. The script is run and, if it fails, its output is sent back to the model, allowing up to three further attempts. Under the *constrained agent*, the model runs its own code, reads the output, revises as many times as it needs, and decides when it is done. We use a simpler agent than the ones widely available, with only a shell, no web search, and instructions to use only the task data and the files the model creates inside a folder made for that run.

We also run every task in all three languages (Stata, R, and Python) and with two different prompts. One prompt is a simpler *zero-shot* version that only has a statement of what the task requires. The other version is *one-shot* (Brown et al., 2020) and has a related code example. We study every combination with six LLMs: Anthropic's Claude Sonnet 4.6, Haiku 5.5, and Opus 5.5 through the Claude Code harness and OpenAI's GPT-5.6 Luna, GPT-6 Luna, and GPT-6 Sol through the Codex harness.[1] They come from the two providers with the largest shares of enterprise spending on coding with language models (Tully et al., 2025).

We created 40 tasks about topics in statistics and econometrics. The tasks cover linear regression, classical tests, generalized linear models, difference-in-differences, instrumental variables, regression discontinuity, survival analysis, censored and truncated regression, time series, decomposition, and discrete choice. Each of them has an associated dataset and reference code that does the estimation being asked about. A run is one attempt by one model at one task. Each run gives a model the prompt to handle that dataset and produce an estimate, and that result is compared to the reference. Success means that the scored values (the estimates and, in some tasks, the standard errors that the task names) computed by the LLM-generated code match the ones generated by the reference within a tolerance and are closer to them than to the values of a known wrong method.

[1] A harness is the software that passes the task to a model, runs the commands the model requests, and returns their output to it.

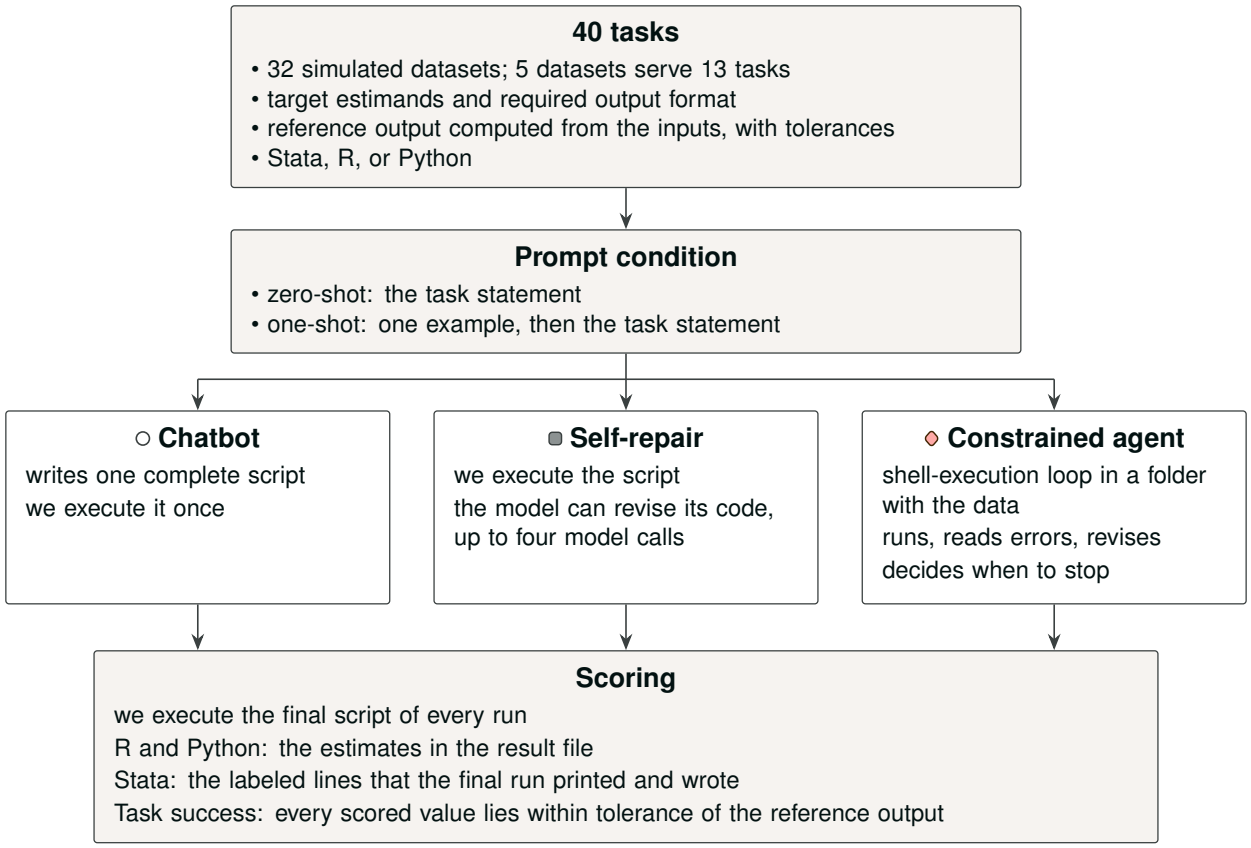


Figure 1: Design of the Benchmark

Notes: Each of the 40 tasks gives a dataset, the target estimands, and a reference output. Every model runs each task in three languages, with two prompts and three agency conditions, ten times in each combination, and each run is scored against the reference output within the task tolerances.

We estimate a regression model with task fixed effects that lets us calculate the effect of agency, prompting, and language on task success. We observe that more agency improves task success in all six models, as is expected. In particular, it rises from 83.7 percent under the chatbot to 93.3 percent under self-repair and 97.4 percent in the most agentic condition.

The one-shot example's effect on chatbot task success is 4.4 percentage points. This effect falls with agency to 0.8 points under the constrained agent. Averaged over agency and language, the example adds 2.3 points.

The gap between Stata and Python declines with agency. This gap is 24.1 percentage points under the zero-shot chatbot, with task success lower in Stata than in Python. However, that gap under the zero-shot constrained agent falls to 4.6 points.

We then study the types of failures. We define a failure as *loud* when the script does not run or does not return every scored value, and as *silent* when the script runs and returns its output but at least one scored value is wrong. Most chatbot failures are loud. The constrained agent lowers failures from 16.3 to 2.6 per 100 runs, removing most loud failures and reducing silent failures from 4.5 to 2.3. Silent failures fall by 69 to 94 percent for four models, but only by 8 percent for GPT-5.6 Luna and 29 percent for GPT-6 Luna. Most remaining failures are silent. The example mainly reduces loud failures. The share of failures removed by the example remains similar under the constrained agent and the chatbot, 26 against 24 percent.

Agency also raises costs. On average, a constrained-agent run costs 3.4 times as much as a chatbot run. In absolute terms, the runs are cheap. In particular, each additional successful run under the constrained agent costs between $0.02 and $1.13, depending on the model. The constrained agent is also slower, with a median runtime 3.8 times that of the chatbot.

Section I discusses the related literature, Section II describes the design, Section III reports task success and the types of failures, Section IV reports cost and runtime, and Section V concludes.

I Related Literature

Our study relates to research on the reproducibility of empirical work and on how its code is written and checked. Published results cannot always be reproduced with the authors' own code and data, and this code often contains errors (Galiani, Gertler and Romero, 2017; Brodeur et al., 2026*b*). When Stodden, Seiler and Ma (2018) asked the authors of articles in *Science* for their data and code, 44 percent provided them, and the findings of just 26 percent of the articles could be reproduced. Coding errors are harder to notice when the results still look as expected. In a randomized experiment with candidates for research positions who had basic regression skills, a coding error was almost 20 percent more likely to be detected when its result was unexpected (Ferman and Finamor, 2025). AI agents can make the same kind of error, writing code that runs and looks correct at first sight but rests on a plausible yet wrong reading of the task (Stoisser et al., 2026). When AI systems automate machine-learning research, they can change the data and report results selectively without the final manuscript showing it (Luo, Kasirzadeh and Shah, 2026).

AI systems increasingly help researchers review the literature, write code, and build datasets (Korinek, 2025; Afonso et al., 2026). By 2025 an estimated 57 percent of the articles from four large publishers showed evidence of LLM influence in their text (Siler, 2026). Generative AI models can now turn text and images into data for causal estimates (Imai and Nakamura, 2026). Some studies give agents the authors' code and data and ask them to reproduce published results or to judge whether these results are reproducible (Hu et al., 2025; Siegel et al., 2025). Numerical differences in results are large among teams of researchers that all test the same hypothesis on the same data (Breznau et al., 2022). Other studies give the same empirical question to several AI agents and measure the dispersion of their estimates (Grundl, 2026; Gao and Xiao, 2026; Huang, Menkveld and Yu, 2026; Bertran, Fogliato and Wu, 2026). We depart from these studies by defining the method and the target output in each task instead of allowing agents to choose the specification. We score every run against a reference output computed from simulated data.

Existing benchmarks for code written by language models follow two approaches: some run automated tests on the code or on the environment it produces (Chen et al., 2021; Lai et al., 2023; Cassano et al., 2023; Jimenez et al., 2024), and others score the model's work with human experts, rubrics, answer keys, or an LLM judge (Starace et al., 2025; Song et al., 2026; Zhu et al., 2026). Like CausalVerify (Zhang et al., 2026), we run model-written code on simulated data and compare estimates with a reference estimator. CausalVerify, the closest benchmark to ours, covers four causal designs in R. In CausalVerify, 15.5 percent of the workflows that ran returned an incorrect estimate or no target estimate when each of seven models wrote its code in one attempt. In economics, studies have asked an LLM to write code that replicates the analyses of published papers (Song, Wu and Zhou, 2025), built an agent that calls a library of econometric functions (Chen et al., 2025), and compared the accuracy and cost of models on Stata tasks with exact numerical answers (El-Tokhy, 2026).

Stata predominates in research code in economics. Stata appears in 73 percent of 2,552 replication packages of the American Economic Association journals, with R in 4 percent and Python in 2 percent (Vilhuber, Turitto and Welch, 2020). Econometric packages can return different numbers for the same estimator (McCullough and Vinod, 1999) and have their own programming conventions. Cox (2005) shows the latter for Stata. Using coding agents as assistance for researchers may help to reduce the cost of working in an unfamiliar language. Among software developers on GitHub, the adoption of Claude Code is associated with commits in more programming languages, including languages absent from their earlier observed public commits (Quispe and Xu, 2026).

Mizrahi et al. (2024) suggest that evaluating models on several prompts is necessary because results can change with prompt wording. In R and Racket, two languages with little

training data, in-context examples can raise the code-generation performance of models that write code without running it (Giagnorio, Martin-Lopez and Bavota, 2025).

In an agent framework, a model works through a sequence of actions and observations (Yao et al., 2023; Yang et al., 2024). Relatedly, self-debugging and similar techniques let a model update its output in response to tests or critiques (Chen et al., 2024; Olausson et al., 2024). Yao et al. (2026) show that the harness alters completion, efficiency, and failure behavior so much that one should report capability for each model and harness pair. For example, when fixing injected errors in R scripts drawn from five published papers, Qwen3-Coder recovers the original results in 69 to 96 percent of the test cases with an agent workflow, compared with 53 to 55 percent with a full-context prompt workflow (Shah, Hopfgartner and Bleier, 2026).

In those repair and reproduction tasks, the model begins with existing code or a published paper. Instead, our design requires every run to write new econometric code from a task statement. In addition, we report cost alongside accuracy, following the recommendation of Kapoor et al. (2025) for agent evaluations.

II Design

Our six models each attempt 40 econometric tasks across all combinations of language (Stata, R, or Python), prompt (zero-shot or one-shot), and agency condition (chatbot, self-repair, or constrained agent). A design cell is one combination of task, language, prompt, and agency condition (Figure 1). We run each cell ten times per model, setting Claude Sonnet 4.6 to medium effort and the other five models to low reasoning effort (Appendix A1). Because neither harness exposes a seed or a temperature, all six models run at their provider's default sampling settings. The resulting analysis sample contains 7,200 runs per model, for a total of 43,200 runs. Appendix A1 also gives the versions of Stata, R, Python, and their packages.

A Degree of agency

Meng et al. (2026) list the functions that an agent harness performs. Four of these functions differ across the three agency conditions: the loop of model calls and the rule that ends it, the tools the model may call, the context it receives, and the record the run leaves behind.

Specifically, under the *chatbot* condition (a single model call) we obtain one complete script, execute it once, and return nothing to the model, as in a chat window where the researcher pastes a request and runs the script that comes back. Under *self-repair*, we then send a repair request (when the script fails or its output lacks a required value) with the

previous script, the end of the captured run output, and a list of the missing values, and the model may submit up to three revised scripts. In the *constrained agent* condition, the model executes shell commands in a per-run working folder that holds the input data. In particular, it writes, runs, and revises its script as often as it chooses and declares when it is finished.

Every constrained-agent prompt also instructs the model to verify its calculation before it finishes, with at least one independent numerical check, and to save a short account of the checks. The contrast between the constrained agent and the chatbot therefore also includes the effect of this instruction. In this setup, the condition is constrained in both its tools (because the model has only a shell and no web-search tool) and its sources (since the prompt restricts it to the task statement, the supplied data, the installed software, and the files it creates).[2]

B Tasks and scoring

Five elements were specified for each task: a dataset, target estimands, a required output format, a reference output computed in advance, and numerical tolerances (Appendices A2 and A4). The dataset of each task is simulated, and this dataset and the reference output are kept the same in all its cells and repetitions. The prompt specifies the language, the dataset and its variables, the quantity to estimate and the sample, the required estimator and inference convention, and the required output. R and Python scripts write a result file, while Stata scripts print one labeled `ANSWER:` line for each estimate. The example is the only difference between the two prompt conditions, with the one-shot prompt placing it before the task statement. This example contains a short statement of a related task, complete code in the target language on separate data, and the output conventions required by the scored task (Appendix A3).

We center our attention on the primary outcome, task success, a binary outcome in which a run succeeds when its output reports every scored value within tolerance of the reference output and closer to it than to the value of a known wrong method. A point estimate matches its reference if its deviation is within 1 percent or 0.005, whichever is larger. A scored standard error must match the reference or an accepted alternative convention within a relative tolerance of at most 1 percent (Appendix A4). We score the output of the execution of the final script of every run. We classify failures by whether the script returns its output (see Section III.C).

[2] Appendix A1 records, for each harness, the tools it authorized and the time limits it applied.

C Cost and runtime measures

In our measurement, the cost of a run is the price of the tokens consumed by its model calls, at the fixed list price of each token type (Appendix C). In particular, it covers every attempt of a self-repair run and every model turn of a constrained-agent session. The mean cost over the runs of a condition serves as a measure of cost per run. As a complement, cost per successful run divides the total cost of those runs by the number of successes among them. We define runtime as the wall-clock duration of a run's calls to the model provider, which includes script executions inside a constrained-agent session but excludes our subsequent execution of the final script.

D Analysis and inference

We estimate the effects of the design dimensions in a regression model, which treats each run as one observation and allows each task its own intercept,

$$\text{Success}_{ir} = \tau_i + X'_{ir}\beta + \varepsilon_{ir}, \tag{1}$$

where Success_{ir} equals one if run r of task i succeeds and zero otherwise, τ_i is the task fixed effect, and X_{ir} holds indicators for every combination of prompt, agency condition, and language in the $2\times3\times3$ design, with zero-shot, chatbot, and Python as the omitted categories. Incorporating all interactions among the three dimensions (Angrist and Pischke, 2009) in X_{ir}, the regression reproduces the observed success rate of each of the 18 combinations of prompt, agency condition, and language, since every cell holds ten runs for each model. Every estimate we report is a difference between two such rates or between averages of them. Thus, the regression organizes the cell comparisons and supplies their standard errors.

We then estimate equation (1) separately for each model and once on the six models pooled with a fixed effect for every pair of model and task and with the same β for the six models. In our design, an equal contribution of ten runs to every cell produces a pooled estimate that equals the simple average of the six per-model estimates. A second pooled specification with a separate β for each model tests whether an estimate differs between the six models.

The language contrasts in Table 1 compare R and Stata with Python in zero-shot chatbot runs, the baseline condition. Two more compare Stata with Python over the whole design and under the zero-shot constrained agent. The prompt and agency contrasts each average over the dimensions their rows do not name. In our design, the interaction contrast measures

how the one-shot effect changes under the constrained agent relative to the chatbot, averaged over the three languages. Its rows in Table 1 also report the change under self-repair.

Runs of a given task use the same dataset and face the same difficulty, and runs within the same cell are correlated. In particular, the within-cell intraclass correlation for task success ranges from 0.40 for Claude Haiku 5.5 to 0.59 for Claude Sonnet 4.6. Because each cell belongs to a single task, we cluster standard errors by task to account for both sources of dependence, and we assess statistical significance at the 5 percent level using these standard errors and a t distribution with 39 degrees of freedom. However, 13 of the 40 tasks draw on five common datasets, so clustering by task does not capture the cross-task dependence induced by shared data (Appendix A2). We therefore also repeat the inference in Appendix D by clustering on the 32 input datasets, and every contrast repeated there remains significant. Every bootstrap interval is a percentile interval.

We ran the analysis in Python 3.14.6 with NumPy 2.4.2, pandas 3.0.0, SciPy 1.18.0, statsmodels 0.14.6, and matplotlib 3.11.0.

III Results

Unless stated otherwise, the rates and estimates below cover all runs of all six models. Task success is 91.5 percent in the whole sample, ranging from 84.0 percent for GPT-6 Luna to 98.2 percent for Claude Opus 5.5.

A Agency effects

We find that, with increasing agency, task success rises for all six models. In particular, task success is 83.7 percent under the chatbot, 93.3 percent under self-repair, and 97.4 percent under the constrained agent. Table 1 reports the differences from the chatbot as the agency contrasts of equation (1), which yield gains of 9.6 percentage points for self-repair and 13.7 points for the constrained agent. The constrained agent's gain differs across the six models ($p < 0.001$). It is largest for GPT-6 Luna, at 20.3 points, and smallest for Claude Opus 5.5, at 2.9 points, since Claude Opus 5.5 already succeeds in 96.6 percent of chatbot runs. Under the constrained agent, task success ranges from 93.1 percent for GPT-6 Luna to 99.8 percent for GPT-6 Sol, against 72.8 to 96.6 percent under the chatbot.

Self-repair alone achieves 70 percent of the constrained agent's improvement. Most self-repair runs revise at most once (Appendix A1). This suggests that no more than three revisions of a failed or incomplete script achieve most of the gain from agency. The constrained agent adds the remainder. A single constrained-agent run succeeds in 97.4 percent

Table 1: Effects of Language, Prompting, and Agency on Task Success

Reasoning effort	Claude Sonnet 4.6 medium (1)	GPT-5.6 Luna low (2)	Claude Haiku 5.5 low (3)	GPT-6 Luna low (4)	Claude Opus 5.5 low (5)	GPT-6 Sol low (6)	Total (pooled) (7)	Equal across models (p-value) (8)
Agency, relative to the chatbot								
Self-repair	12.1	13.0	9.3	13.3	2.0	7.8	9.6	<0.001
(revises on error or missing value)	(1.7)	(1.5)	(1.4)	(1.7)	(0.7)	(1.6)	(1.0)	
Constrained agent	18.2	18.2	11.9	20.3	2.9	10.6	13.7	<0.001
(runs and revises its script)	(2.0)	(2.1)	(1.9)	(2.4)	(1.0)	(1.7)	(1.3)	
Prompt, one-shot relative to zero-shot								
One example in the prompt	2.3	4.1	1.8	5.0	−0.2	1.0	2.3	0.015
	(1.6)	(1.6)	(0.7)	(1.8)	(0.4)	(0.6)	(0.8)	
One example, chatbot	5.3	7.0	2.9	8.8	0.0	2.5	4.4	0.008
	(3.4)	(2.5)	(1.5)	(2.9)	(0.9)	(1.5)	(1.5)	
One example, self-repair	0.9	3.8	1.8	3.8	−0.1	0.5	1.8	0.18
	(1.6)	(2.0)	(0.9)	(2.3)	(0.4)†	(0.8)	(0.9)	
One example, constrained agent	0.5	1.5	0.7	2.5	−0.5	0.1	0.8	0.024
	(0.6)	(1.4)	(0.3)	(1.1)	(0.3)	(0.1)	(0.4)	
Language, relative to Python								
R, zero-shot chatbot	3.3	−1.5	−1.8	6.5	−1.0	−5.8	0.0	0.15
	(4.9)	(3.3)	(2.8)	(4.5)	(1.0)†	(2.4)	(1.7)	
Stata, zero-shot chatbot	−25.8	−33.0	−22.3	−35.3	−9.3	−19.3	−24.1	0.002
	(7.0)	(5.9)	(5.6)	(5.8)	(3.3)	(4.8)	(3.9)	
Stata, average over prompts and agency	−11.4	−18.0	−8.6	−20.4	−4.7	−7.8	−11.8	<0.001
	(2.7)	(3.5)	(2.1)	(3.4)	(2.0)	(2.3)	(2.0)	
Stata, zero-shot constrained agent	−2.3	−9.3	−2.3	−13.3	−0.8	0.3	−4.6	0.001
	(1.4)	(3.4)	(1.0)	(3.5)	(0.8)†	(0.4)	(1.3)	
One-shot effect, change from the chatbot								
Under self-repair	−4.4	−3.3	−1.2	−4.9	−0.1	−2.0	−2.6	0.28
	(2.7)	(2.1)	(1.5)	(2.4)	(0.8)	(1.6)	(1.1)	
Under the constrained agent	−4.8	−5.5	−2.3	−6.3	−0.5	−2.4	−3.6	0.11
	(3.3)	(2.5)	(1.5)	(2.7)	(0.9)	(1.6)	(1.3)	
Task-success rate (percent)								
Chatbot, all languages and prompts	79.9	76.0	87.7	72.8	96.6	89.2	83.7	
Constrained agent, all languages and prompts	98.1	94.2	99.6	93.1	99.5	99.8	97.4	
Zero-shot chatbot writing Python	84.8	84.0	94.3	78.0	100.0	96.3	89.5	
Runs	7,200	7,200	7,200	7,200	7,200	7,200	43,200	
Tasks (clusters)	40	40	40	40	40	40	40	
Task fixed effects	Yes	Yes	Yes	Yes	Yes	Yes	Model by task	

Notes: Effects are differences in task-success rates, in percentage points, from equation (1) with task fixed effects (model-by-task fixed effects in the pooled column 7), averaged over the unnamed design dimensions. Two reference levels are used: the chatbot rate over all languages and prompts for the agency rows and the zero-shot chatbot rate in Python for the first two language rows. The prompt rows compare one-shot with zero-shot runs, and the last two language rows compare Stata with Python, in the conditions each row names. Standard errors clustered by task (40 clusters) are in parentheses. Column 8 gives the p-value of an F test, with 5 and 39 degrees of freedom and standard errors clustered by task, that the six model estimates in the row are equal. In the rows on the change in the one-shot effect, a negative change means that the example adds less with agency. Reasoning effort, in the header, is the provider setting for how much the model reasons before it answers. A dagger means that fewer than three tasks have different rates in the two compared cells. Table D2 gives wild cluster bootstrap and Holm-adjusted p-values for every entry.

of cases. To see whether repeated chatbot runs can match this rate, we compute the chatbot's pass@10, the probability that at least one of ten runs on the same task, language, and prompt succeeds. This probability is 95.9 percent, so, pooled over the six models, a single constrained-agent run succeeds at least as often as the best of ten chatbot runs (Appendix C3).

B Prompt and language effects

The one-shot example raises task success under the chatbot but adds little under the constrained agent. In particular, it increases task success, averaged over the three languages, by 4.4 percentage points under the chatbot, 1.8 under self-repair, and 0.8 under the constrained agent. In Panel A of Figure 2, the one-shot and zero-shot lines almost coincide under the constrained agent.

The prompt contrast of Table 1, the average effect of the example over the three agency conditions, is 2.3 percentage points. The interaction contrast, the change in this effect from the chatbot to the constrained agent, is −3.6 points (Table 1, rows on the change in the one-shot effect). The negative sign means that the example adds less when the model can run its code. However, the example removes a similar share of zero-shot failures under the chatbot and under the constrained agent, 24 and 26 percent. Near the ceiling, the smaller effect therefore does not necessarily indicate that the example and the constrained agent are substitutes (Appendix D).

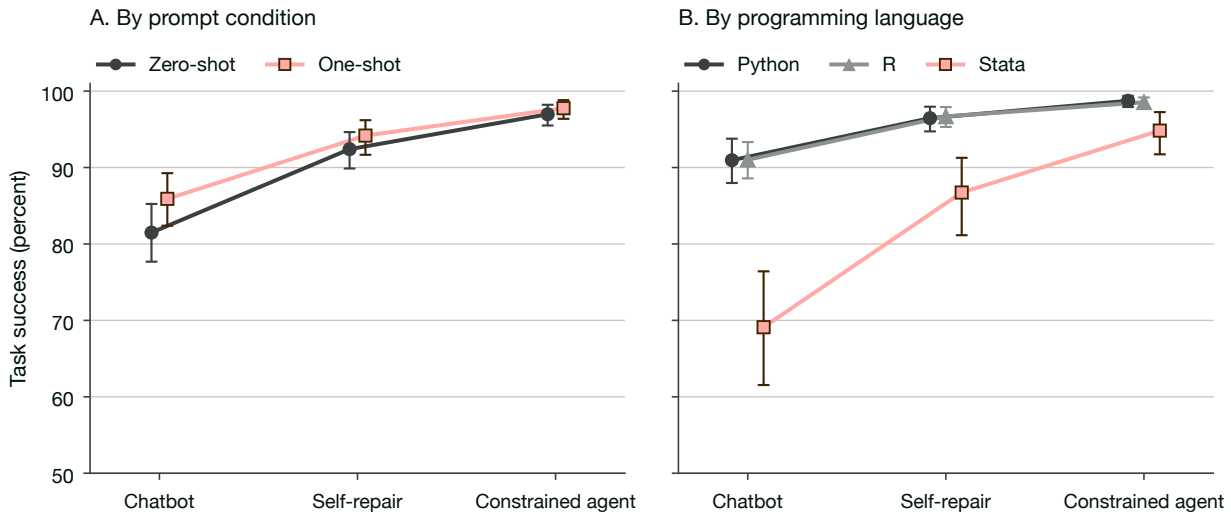


Figure 2: Task Success by Agency Condition

Notes: Two panels are shown: the two prompt conditions in Panel A and the three languages in Panel B. Each point covers 7,200 runs in Panel A and 4,800 runs in Panel B. Points pool the six models, the three languages in Panel A, and the two prompt conditions in Panel B. Whiskers show 95 percent intervals from a cluster bootstrap over the 40 tasks with 4,000 draws.

Language choice shows the same dependence on agency. Under the zero-shot chatbot (the baseline condition), Stata success rates are 24.1 percentage points below Python, and R success rates are about equal to those of Python. We find that, with increasing agency,

the Stata gap narrows for all six models (Table 1, and Panel B of Figure 2 pooled over prompts). The success rate of zero-shot Stata runs rises from 65.4 percent under the chatbot to 94.0 percent under the constrained agent, while the gap with Python falls to 4.6 points. Across the six models, this gap is largest for GPT-6 Luna, at 13.3 points, and below zero for GPT-6 Sol, whose Stata runs succeed slightly more often than its Python runs. As shown in Section III.C, the Stata gap under the chatbot comes mainly from loud failures. The prompt and the language both play a smaller role in determining task success once the model can run and revise its code.

C Failures

We classify a failure as loud if the script fails to execute or fails to produce complete output, leaving the user with an error message or a missing result. In contrast, a failure is silent if the script runs to completion and returns output, but at least one scored value is wrong (Figure 3 and Appendix B). Moving from the chatbot to the constrained agent reduces loud failures from 11.8 to 0.3 per 100 runs, while silent failures decline from 4.5 to 2.3 per 100 runs. Consequently, the overall failure rate drops from 16.3 to 2.6 per 100 runs, with most remaining failures silent. Under self-repair, silent failures instead rise to 5.5 per 100 runs, because a revision can turn a loud failure into a silent one (Appendix B). Silent failures fall unevenly across models, by 69 percent from 6.2 to 1.9 per 100 runs for Claude Sonnet 4.6 (and by 69 to 94 percent for Claude Haiku 5.5, Claude Opus 5.5, and GPT-6 Sol), but only from 6.4 to 5.8 for GPT-5.6 Luna and from 7.7 to 5.5 for GPT-6 Luna (Appendix B). These two models account for 80 percent of the silent failures that remain under the constrained agent.

This split between loud and silent failures is consistent with the two patterns of Section III.B, the falling effect of the example and the closing of the language gap. In particular, the one-shot example reduces mainly loud failures under the chatbot, from 13.5 to 10.0 per 100 runs, while silent failures fall from 5.0 to 4.1. The example therefore raises task success mainly by making more scripts run and return their output. The constrained agent leaves few loud failures for the example to prevent. The languages differ mainly in loud failures. Under the chatbot, 23.3 percent of Stata scripts fail to run or return complete output, while the shares are 5.9 percent for Python and 6.1 percent for R. However, the shares with a wrong scored value differ far less across the three languages (Table B1). We find that, under the constrained agent, those loud failures decline in the three languages for all six models, and most of the Stata gap closes with them. What remains is mostly silent. Under the

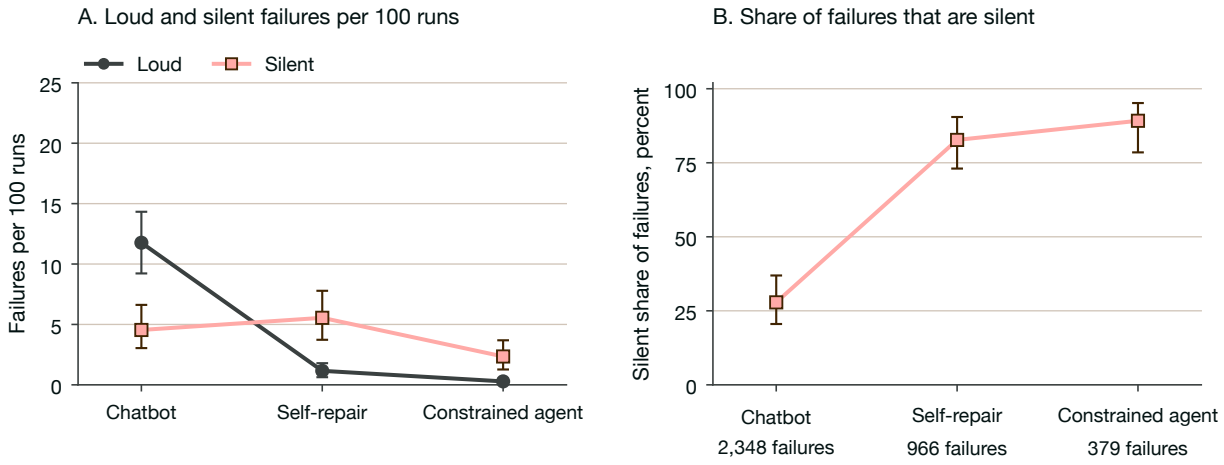


Figure 3: Failure Types by Agency Condition

Notes: Two failure types are reported: execution and output failures (loud) and incorrect scored values (silent) per 100 runs in Panel A. Two measures of these failures are reported in Panel B: their silent share and their number in each condition on the second line of each tick label. Each point covers the 14,400 runs of all six models, both prompt conditions, and the three languages. We derive whiskers as 95 percent intervals, using a cluster bootstrap over the 40 tasks with 4,000 draws. Table B1 also reports the rates for each model.

zero-shot constrained agent, 5.0 percent of Stata runs have a wrong scored value, against 1.3 percent of Python runs.

IV Cost and Runtime

Unless stated otherwise, the numbers cover both prompt conditions.

Mean cost per run is 235 percent higher (from 1.46 cents under the chatbot to 4.90 cents) if the run is under the constrained agent, since the constrained agent adds model calls. For Claude Sonnet 4.6, the most expensive of the six models, the mean cost per run rises from 4.56 to 14.09 cents, and for GPT-6 Luna, the cheapest overall, from 0.0768 to 0.482 cents. Cost per successful run, in contrast, rises less, by 188 percent, from 1.75 to 5.03 cents, because the conditions that cost more also produce more successes. We find that self-repair accounts for about 10 percent of the increase in cost and, as Section III.A shows, for 70 percent of the gain in task success. Claude Sonnet 4.6 runs cost 59.4 times as much as GPT-6 Luna runs under the chatbot, while they cost 29.2 times as much under the constrained agent (Figure C1). Across the six models, a constrained-agent run costs from 2.4 times (Claude Opus 5.5) to 6.3 times (GPT-6 Luna) as much as a chatbot run. The constrained agent

costs more per run than the chatbot and succeeds more often in every language and for all six models, and the gains in task success are largest in Stata (Figure 4 and Figure C1). The constrained agent of the three cheaper models (GPT-5.6 Luna, Claude Haiku 5.5, and GPT-6 Luna) succeeds in 95.6 percent of runs. The chatbot of the three more expensive models succeeds in 88.6 percent and costs 4.6 times as much per run.

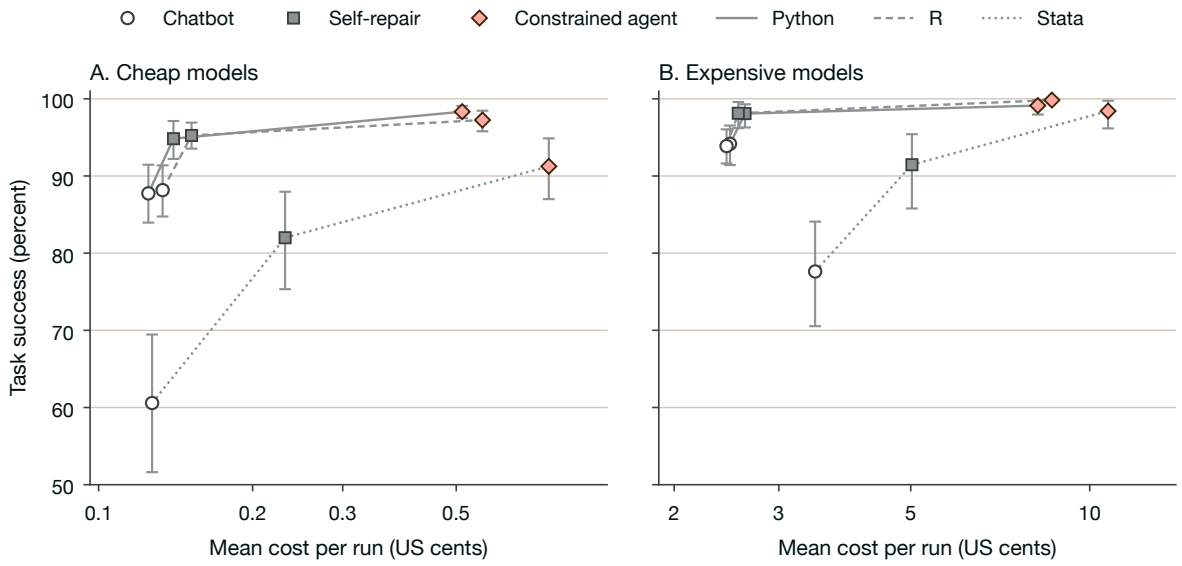


Figure 4: Task Success and Mean Cost per Run Across Languages and Agency Conditions

Notes: Panel A shows the cheap models, the three with the lowest mean cost per run (GPT-5.6 Luna, Claude Haiku 5.5, and GPT-6 Luna), and Panel B the expensive models (Claude Sonnet 4.6, Claude Opus 5.5, and GPT-6 Sol). Within each panel, a line connects the chatbot (open circle), self-repair (gray square), and constrained agent (pink diamond) for a given language, and the line style marks the language. Every point pools the three models and both prompt conditions over 2,400 runs. Both panels share a common task-success axis but use separate logarithmic axes for cost. Whiskers show 95 percent intervals for task success from a cluster bootstrap over the 40 tasks with 4,000 draws.

Providing a one-shot example reduces the mean cost per run by 14 percent. Although the example makes every prompt longer, it produces shorter responses, and output tokens carry the highest price among token types. Under self-repair, the example further reduces costs by preventing loud failures and the subsequent revision turns they trigger.

For five of the six models, Stata is the costliest language. A run in Stata costs 1.4 times the average cost of a Python run, while a successful Stata run costs 1.6 times as much because Stata runs succeed less often (Table 1).

We measure the incremental cost of agency as the change in mean cost per run divided by the change in task success when moving from the chatbot to the constrained agent. The constrained agent increases task success by 13.7 percentage points and raises mean cost per

run by 3.4 cents, which implies a price of about $0.25 for each additional successful run, pooled over the six models. By model, this price ranges from $0.02 for GPT-6 Luna to $1.13 for Claude Opus 5.5, whose chatbot already succeeds in 96.6 percent of runs.

The median runtime under the constrained agent is 3.8 times that of the chatbot, taking 34.8 versus 9.2 seconds (Figure C3). By contrast, self-repair increases median runtime by just 10 percent, because most self-repair runs involve no revision.

V Conclusion

We evaluate how frequently large language models produce code that recovers the target estimates once the researcher has fixed the research question, the estimation method, and the identification strategy. We find that the agency condition matters more for task success than either the prompt design or the language. Under the chatbot, loud failures predominate, accounting for most of the advantage of the one-shot example as well as the performance deficit of Stata relative to R and Python. Allowing the model to execute and revise its own code eliminates most chatbot failures, and most of this improvement can be achieved with up to three revisions of an incomplete or broken script. Because agency primarily eliminates loud failures, little room remains for the one-shot example to help, and the gap across languages largely disappears.

Nevertheless, a script that executes without error and outputs numerical estimates can still be substantively incorrect. While the constrained agent removes most loud failures, it removes only 48 percent of silent failures, a share that ranges from 8 percent for GPT-5.6 Luna to 94 percent for Claude Haiku 5.5. Most remaining failures are therefore silent, and introducing agency shifts the nature of the errors that a researcher must detect. As researchers move from chat interfaces to coding agents (Massenkoff et al., 2026), they will face fewer errors overall, but a greater proportion of the remaining errors will be silent.

Our findings are specific to the six models and their two harnesses, the 40 tasks, the English-language prompts, the three languages, and the three agency conditions in our study. However, researchers can rerun this benchmark as new models and agent harnesses emerge. A separate question, which our benchmark leaves aside and which recent papers study using real data (Grundl, 2026; Gao and Xiao, 2026; Huang, Menkveld and Yu, 2026), is how an LLM selects an econometric specification for an applied problem. Across all agency conditions, verifying that the econometric specification and the resulting estimates are correct remains the responsibility of the researcher.

References

Afonso, Santiago, Sebastian Galiani, Ramiro H. Gálvez, and Raul A. Sosa. 2026. "Deep Research on a Loop: Using AI Agents to Construct Economic Datasets." National Bureau of Economic Research Working Paper 35188.

Angrist, Joshua D., and Jörn-Steffen Pischke. 2009. *Mostly Harmless Econometrics: An Empiricist's Companion.* Princeton University Press.

Angrist, Joshua D., and Jörn-Steffen Pischke. 2010. "The Credibility Revolution in Empirical Economics: How Better Research Design Is Taking the Con out of Econometrics." *Journal of Economic Perspectives*, 24(2): 3–30.

Bertran, Martin, Riccardo Fogliato, and Zhiwei Steven Wu. 2026. "Many AI Analysts, One Dataset: Navigating the Agentic Data Science Multiverse." *Proceedings of the National Academy of Sciences*, 123(29): e2606495123.

Breznau, Nate, Eike Mark Rinke, Alexander Wuttke, Hung H. V. Nguyen, Muna Adem, Jule Adriaans, et al. 2022. "Observing Many Researchers Using the Same Data and Hypothesis Reveals a Hidden Universe of Uncertainty." *Proceedings of the National Academy of Sciences*, 119(44): e2203150119.

Brodeur, Abel, David Valenta, Alexandru Marcoci, Juan P. Aparicio, Derek Mikola, Bruno Barbarioli, Rohan Alexander, Lachlan Deer, Tom Stafford, Lars Vilhuber, et al. 2026*a*. "AI-Assisted Teams Outperform AI-Led Teams but Not Human-Only Teams in Assessing Research Reproducibility in Quantitative Social Science." *Proceedings of the National Academy of Sciences*, 123(22): e2524747123.

Brodeur, Abel, Derek Mikola, Nikolai Cook, Lenka Fiala, Thomas Brailey, Ryan Briggs, Alexandra de Gendre, Yannick Dupraz, Jacopo Gabani, Romain Gauriot, et al. 2026*b*. "Reproducibility and Robustness of Economics and Political Science Research." *Nature*, 652(8108): 151–156.

Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. "Language

Models Are Few-Shot Learners." In *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33, 1877–1901.

Cassano, Federico, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q. Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. 2023. "MultiPL-E: A Scalable and Polyglot Approach to Benchmarking Neural Code Generation." *IEEE Transactions on Software Engineering*, 49(7): 3675–3691.

Chatterji, Aaron, Thomas Cunningham, David J. Deming, Zoe Hitzig, Christopher Ong, Carl Yan Shan, and Kevin Wadman. 2025. "How People Use ChatGPT." National Bureau of Economic Research Working Paper 34255.

Chen, Mark, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. "Evaluating Large Language Models Trained on Code." *arXiv preprint arXiv:2107.03374*, Version 2, July 2021.

Chen, Qiang, Tianyang Han, Jin Li, Ye Luo, Zigan Wang, Yuxiao Wu, Xiaowei Zhang, and Tuo Zhou. 2025. "Can AI Master Econometrics? Evidence from Econometrics AI Agent on Expert-Level Tasks." *arXiv preprint arXiv:2506.00856*, Version 3, January 2026.

Chen, Xinyun, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. "Teaching Large Language Models to Self-Debug." In *International Conference on Learning Representations (ICLR)*.

Cox, Nicholas J. 2005. "Suggestions on Stata Programming Style." *Stata Journal*, 5(4): 560–566.

Cui, Kevin Zheyuan, Mert Demirer, Sonia Jaffe, Leon Musolff, Sida Peng, and Tobias Salz. 2026. "The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers." *Management Science*. Published online February 27, 2026.

El-Tokhy, Khaled. 2026. "LLM Stata Benchmark." *Online benchmark, https://khaledeltokhy.com/benchmarks/*, Accessed October 2, 2026.

Ferman, Bruno, and Lucas Finamor. 2025. "There Must Be an Error Here! Experimental Evidence on Coding Errors' Biases." *arXiv preprint arXiv:2508.20069*, Version 2, September 2025.

Galiani, Sebastian, Paul Gertler, and Mauricio Romero. 2017. "Incentives for Replication in Economics." National Bureau of Economic Research Working Paper 23576.

Gao, Ruijiang, and Steven Chong Xiao. 2026. "Nonstandard Errors in AI Agents." *arXiv preprint arXiv:2603.16744*, Version 2, March 2026.

Garg, Prashant, and Thiemo Fetzer. 2025. "Causal Claims in Economics." *arXiv preprint arXiv:2501.06873*, Version 2, February 2026.

Giagnorio, Alessandro, Alberto Martin-Lopez, and Gabriele Bavota. 2025. "Enhancing Code Generation for Low-Resource Languages: No Silver Bullet." In *IEEE/ACM International Conference on Program Comprehension (ICPC)*. 478–488.

Grundl, Serafin. 2026. "A Comparison of Agentic AI Systems and Human Economists." *SSRN Working Paper 6551598*, Board of Governors of the Federal Reserve System. Version of July 9, 2026.

Hu, Chuxuan, 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*. 23616–23626.

Huang, Wenqian, Albert J. Menkveld, and Shihao Yu. 2026. "AI 'Errors'." *SSRN Working Paper 6408138*, Version of September 11, 2026.

Imai, Kosuke, and Kentaro Nakamura. 2026. "Leveraging Generative AI for Causal Inference with Unstructured Data." *Proceedings of the National Academy of Sciences*, 123(36): e2530532123.

Jimenez, Carlos E., John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?" In *International Conference on Learning Representations (ICLR)*.

Johnston, Drew, David Holtz, Alex Martin Richmond, Christopher Ong, Prasanna Tambe, and Aaron Chatterji. 2026. "The Shift to Agentic AI: Evidence from Codex." *arXiv preprint arXiv:2606.26959*.

Kapoor, Sayash, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. 2025. "AI Agents That Matter." *Transactions on Machine Learning Research*.

Korinek, Anton. 2025. "AI Agents for Economic Research." National Bureau of Economic Research Working Paper 34202.

Lai, Yuhang, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. "DS-1000: A Natural and Reliable Benchmark for Data Science Code Generation." In *Proceedings of the 40th International Conference on Machine Learning (ICML)*. 18319–18345.

Luo, Ziming, Atoosa Kasirzadeh, and Nihar B. Shah. 2026. "The More You Automate, the Less You See: Hidden Pitfalls of Autonomous AI Scientists." *Proceedings of the National Academy of Sciences*, 123(41): e2610214123.

Lyttelton, Thomas, Maxim Massenkoff, and Nathan Wilmers. 2026. "Coding Agents in the Social Sciences." *Anthropic Economic Research, May 27, 2026,* https://www.anthropic.com/research/coding-agents-social-sciences.

Massenkoff, Maxim, Eva Lyubich, Szymon Sacher, Zoe Hitzig, Shaoyi Zhang, Ryan Heller, and Peter McCrory. 2026. "Anthropic Economic Index Report: Cadences." *Anthropic,* https://www.anthropic.com/research/economic-index-june-2026-report, Accessed August 24, 2026.

McCullough, B. D., and H. D. Vinod. 1999. "The Numerical Reliability of Econometric Software." *Journal of Economic Literature*, 37(2): 633–665.

Meng, Qianyu, Yanan Wang, Liyi Chen, Yihang Li, Wei Wu, Wenyuan Jiang, Qimeng Wang, Chengqiang Lu, Yan Gao, Yi Wu, and Yao Hu. 2026. "Agent Harness for Large Language Model Agents: A Survey." *Preprints.org, doi:10.20944/preprints202604.0428.v3*.

Mizrahi, Moran, Guy Kaplan, Dan Malkin, Rotem Dror, Dafna Shahaf, and Gabriel Stanovsky. 2024. "State of What Art? A Call for Multi-Prompt LLM Evaluation." *Transactions of the Association for Computational Linguistics*, 12: 933–949.

Olausson, Theo X., Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. 2024. "Is Self-Repair a Silver Bullet for Code Generation?" In *International Conference on Learning Representations (ICLR)*.

Quispe, Alexander, and Kevin Xu. 2026. "Agentic Delegation and the Language Frontier of Software Developers: A Model and Evidence from Claude Code on GitHub." *arXiv preprint arXiv:2605.25438*, Version 2, July 2026.

Shah, Syed Mehtab Hussain, Frank Hopfgartner, and Arnim Bleier. 2026. "Automating Computational Reproducibility in Social Science: Comparing Prompt-Based and Agent-Based Approaches." In *Companion Proceedings of the ACM Web Conference 2026 (WWW Companion '26)*. 989–998.

Siegel, Zachary S., Sayash Kapoor, Nitya Nadgir, Benedikt Stroebl, and Arvind Narayanan. 2025. "CORE-Bench: Fostering the Credibility of Published Research Through a Computational Reproducibility Agent Benchmark." *Transactions on Machine Learning Research*.

Siler, Kyle. 2026. "The Diffusion of Large Language Models in Published Academic Articles." *Proceedings of the National Academy of Sciences*, 123(22): e2605754123.

Song, Xinyi, Lina Lee, Kexin Xie, Xueying Liu, Xinwei Deng, and Yili Hong. 2026. "StatLLM: A Dataset for Evaluating the Performance of Large Language Models in Statistical Analysis." *Scientific Data*, 13: 369.

Song, Ziyu, Shan Wu, and Yuqin Zhou. 2025. "Code Like an Economist: Analyzing LLMs' Code Generation Capabilities in Economics and Finance." *Finance Research Letters*, 86: 108540.

Starace, Giulio, 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." In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*. 56843–56873.

Stodden, Victoria, Jennifer Seiler, and Zhaokun Ma. 2018. "An Empirical Analysis of Journal Policy Effectiveness for Computational Reproducibility." *Proceedings of the National Academy of Sciences*, 115(11): 2584–2589.

Stoisser, Josefa Lia, Marc Boubnovski Martell, Sidsel Boldsen, Kaspar Märtens, and Robert Kitchen. 2026. "Ambig-DS: A Benchmark for Task-Framing Ambiguity in Data-Science Agents." *arXiv preprint arXiv:2605.09698*.

Tan, Lin, Chen Liu, Zhenmin Li, Xuanhui Wang, Yuanyuan Zhou, and Chengxiang Zhai. 2014. "Bug Characteristics in Open Source Software." *Empirical Software Engineering*, 19(6): 1665–1705.

Tully, Tim, Joff Redfern, Deedy Das, and Derek Xiao. 2025. "2025: The State of Generative AI in the Enterprise." *Menlo Ventures, https://menlovc.com/perspective/2025-the-state-of-generative-ai-in-the-enterprise/*, Accessed August 24, 2026.

Vilhuber, Lars, James Turitto, and Keesler Welch. 2020. "Report by the AEA Data Editor." *AEA Papers and Proceedings*, 110: 764–775.

Yang, John, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. "SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering." In *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 37, 50528–50652.

Yao, Shunyu, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. "ReAct: Synergizing Reasoning and Acting in Language Models." In *International Conference on Learning Representations (ICLR)*.

Yao, Yilun, Xinyu Tan, Chao-Hsuan Liu, Yaoming Li, Zhengyang Wang, Wenhan Yu, Zhewen Tan, Yuxuan Tian, Guangxiang Zhao, Lin Sun, Xiangzheng Zhang, and Tong Yang. 2026. "Harness-Bench: Measuring Harness Effects across Models in Realistic Agent Workflows." *arXiv preprint arXiv:2605.27922*.

Zhang, Yonghong, Ricardo Correia, Isabel M. Parra, and Yong Xie. 2026. "CausalVerify: End-to-End Verification of Causal Analyses by Language Models." *arXiv preprint arXiv:2609.07944*, Version 2, September 2026.

Zhu, Youxin, Yixuan Ding, Peng Lai, Longyue Wang, Bingyi Jing, and Guanhua Chen. 2026. "StatABench: Dataset and Framework for Evaluating Statistical Analysis Capabilities of LLMs." *arXiv preprint arXiv:2606.22977*.