Scaling Reproducibility: An AI-Assisted Workflow for Large-Scale Reanalysis[*]

Yiqing Xu[†] (Stanford)   Leo Yang Yang[‡] (HKBU)

February 17, 2026

Abstract

Reproducibility is central to research credibility, yet large-scale reanalysis of empricial data remains costly because replication packages vary widely in structure, software environment, and documentation. We develop and evaluate an agentic AI workflow that addresses this execution bottleneck while preserving scientific rigor. The system separates scientific reasoning from computational execution: researchers design fixed diagnostic templates, and the workflow automates the acquisition, harmonization, and execution of replication materials using pre-specified, version-controlled code. A structured knowledge layer records resolved failure patterns, enabling adaptation across heterogeneous studies while keeping each pipeline version transparent and stable. We evaluate this workflow on 92 instrumental variable (IV) studies, including 67 with manually verified reproducible 2SLS estimates and 25 newly published IV studies under identical criteria. For each paper, we analyze up to three two-stage least squares (2SLS) specifications, totaling 215. Across the 92 papers, the system achieves 87% end-to-end success overall. Conditional on accessible data and code, reproducibility is 100% at both the paper and specification levels. The framework substantially lowers the cost of executing established empirical protocols and can be adapted in empirical settings where analytic templates and norms of transparency are well established.

Keywords: reproducibility, replication, research transparency, open science, AI-assisted workflows, agentic AI, Claude Skills, causal inference

[*]The authors used Claude Code and ChatGPT as research and writing assistants in preparing this manuscript. All interpretations, conclusions, and any errors remain solely the responsibility of the authors.

[†]Yiqing Xu, Assistant Professor, Department of Political Science, Stanford University. Email: yiqingxu@stanford.edu.

[‡]Leo Yang Yang, Research Assistant Professor, Department of Accountancy, Economics and Finance, School of Business, Hong Kong Baptist University, Kowloon, Hong Kong SAR. Email: leoyang@hkbu.edu.hk.

1. Introduction

Reproducibility is fundamental to research credibility and cumulative scientific progress. In empirical social science, reproducible analyses allow researchers to verify published claims, scrutinize identifying assumptions, and assess the practical relevance of new methodological developments. As empirical methods evolve rapidly, access to real-world data and code has become increasingly important not only for assessing research credibility, but also for advancing methodology through systematic reanalysis of existing studies.

Institutional norms have expanded the availability of replication materials. Leading journals in economics and political science now require authors to post data and code, and some conduct in-house replication checks before publication. Yet availability alone does not ensure reproducibility at scale. Replication packages vary widely in software environment, directory structure, naming conventions, documentation quality, and execution logic. Even when materials are public, reproducing results across many papers remains costly and fragile. The bottleneck is operational: executing idiosyncratic replication materials in a standardized and auditable manner requires substantial researcher time.

This paper develops and evaluates an agentic AI workflow to address this execution bottleneck. The workflow combines adaptive coordination with deterministic computation. A large language model (LLM) routes tasks across modular agents that ingest replication materials, identify specifications, reconstruct computational environments, execute models, and generate standardized diagnostic reports. A structured knowledge layer records previously resolved failure patterns and clarifies stage-level responsibilities, allowing the system to accumulate experience across studies while keeping each pipeline version transparent and stable. All numerical operations—data preparation, estimation, and diagnostic computation—are carried out by version-controlled program code. For a fixed pipeline version and fixed inputs, reruns produce identical numerical outputs and retain a complete audit trail of intermediate artifacts and logs. The paper does not propose new estimators or diagnostics.

Instead, we ask whether established empirical protocols can be executed reliably and at scale under real-world conditions. Our evidence suggests that they can.

A central design principle of this workflow is the separation of scientific reasoning from computational execution. Human researchers design diagnostic templates that specify estimands, estimators, robustness checks, and summary measures appropriate for a given research design. Once these templates are fixed, reproduction largely consists of execution-oriented tasks: acquiring replication packages, reconstructing computational environments, locating and running prespecified specifications, extracting analysis datasets, and harmonizing outputs. At the current stage of development, AI systems can not design diagnostic tools that meet the precision standards implied by econometric and statistical theory. We therefore treat diagnostics as human-designed inputs and evaluate whether AI can execute them reliably and reproducibly at scale. This division of labor may evolve as AI systems improve, but it aligns with current research needs.

We evaluate the workflow on a corpus of 92 studies with instrumental variable (IV) designs. Of these, 67 were previously analyzed in Lal et al. (2024), where the authors manually verified the reproducibility of at least one the two-stage least squares (2SLS) coefficient in each study. We extend the analysis to 25 additional IV studies published after the original sample, applying identical inclusion criteria and the same diagnostic template. Across the combined corpus, the workflow targets up to three 2SLS specifications per paper. Each specification corresponds to a model defined by an outcome, a single treatment variable, one or more instruments, and a set of covariates, estimated on a particular sample. In the expanded set of 92 studies, the system achieves a 87% end-to-end reproducibility success. The unsuccessful cases are caused by incomplete replication materials rather than computational instability. Conditional on accessible materials, the pipeline reproduces the benchmark 2SLS estimates exactly and completes all the diagnostic tests.

It is important to note that this level of reliability was not achieved in a single engineering pass. The corpus spans multiple programming languages (mostly Stata and R), estimation commands, directory structures, and idiosyncratic coding practices. Many failure modes arise only when new replication packages are encountered. We therefore adopt an adaptive, human-in-the-loop process. When a recurring failure pattern is identified, it is encoded as a generalized rule in the execution layer and version-controlled between runs. Coverage expands across versions, while numerical behavior remains fixed within each version. The appendix documents the classes of variability encountered and the corresponding adjustments that resolved them.

This adaptive process reflects the challenges we encountered in our prior large-scale reanalysis projects. In earlier work, we manually reconstructed and reanalyzed dozens of published studies using interaction models, IV designs, and parallel trends designs using panel data (Hainmueller, Mummolo and Xu, 2019; Lal et al., 2024; Chiu et al., 2023). Each project required years of coordinated effort. Much of that time was devoted not to methodological development, but to deciphering authors' codebases, repairing path dependencies, reconstructing environments, and harmonizing heterogeneous replication materials. Those experiences revealed that once a diagnostic template is specified, most remaining work is procedural—and therefore, in principle, automatable. They also produced structured benchmark corpora that enable systematic evaluation of an automated executor.

Viewed more broadly, this project connects to long-standing concerns about research credibility. Calls to improve empirical practice—ranging from Leamer's appeal to "take the con out of econometrics" (Leamer, 1983) to the credibility revolution in economics (Angrist and Pischke, 2010) and political science (Torreblanca et al., 2026)—have emphasized transparency, auditability, and disciplined empirical workflows alongside advances in causal identification. More recent discussions of the replication crisis, particularly in psychology (Open Science Collaboration, 2015), highlight the consequences of fragile research pipelines. In this paper, we adopt the now-standard distinction between reproducibility—recovering reported results using the original data and code—and replicability—obtaining similar findings in new studies. Our focus is explicitly on reproducibility. It is not a substitute for replication, but a necessary first step toward credible inference and cumulative research.

The contributions of this paper are threefold. First, we design and implement an adaptive yet version-controlled agentic AI workflow that executes fixed empirical templates across heterogeneous replication materials. Second, we provide systematic evaluation against a manually verified benchmark and a forward extension to newly published studies, documenting a 100% end-to-end success rate in the expanded corpus. Third, we make transparent the variability inherent in real-world replication packages and the engineering adjustments required to accommodate it, offering a disciplined template for similar efforts in other research designs.

The scope of this study is currently limited to empirical social science, particularly causal inference, with data drawn primarily from political science, where norms for sharing replication materials are relatively strong. The workflow depends on the availability of usable code and data and is not designed to recover results when replication materials are missing, incomplete, or fundamentally flawed. The findings therefore speak to what is feasible under current best practices rather than to settings without accessible materials. At the same time, the framework itself is not discipline-specific. Wherever empirical fields have established analytic templates and norms of transparency, the same template–executor approach can be adapted to support large-scale reproducibility and systematic reanalysis.

2. Empirical Corpus and Evaluation Design

This section defines the empirical corpus and evaluation framework used to assess the AI-assisted workflow. The demonstration in this paper focuses exclusively on IV designs. We begin from a manually curated benchmark corpus and then evaluate whether the workflow can (i) reproduce those studies end to end and (ii) extend the analysis to newly published IV studies in the same journals while increasing within-study coverage. We use "data" broadly to include replication packages, code, computational environments, diagnostic templates, and expected outputs.

2.1. Original Benchmark Corpus

The starting point is the corpus of 67 IV studies analyzed in Lal et al. (2024). These studies were drawn from three leading political science journals, *The American Political Science Review* (APSR), *American Journal of Political Science* (AJPS), and *The Journal of Politics* (JOP), published in 2010–2022 and satisfy a common set of design restrictions: linear IV models with a single endogenous regressor (the treatment) and a clearly identified baseline specification. In the original project, we manually selected each paper, located and downloaded replication materials, reconstructed computational environments, reproduced the main results, and applied a prespecified diagnostic template comparing 2SLS and ordinary least squares (OLS) estimates. The key empirical finding is the cross-study 2SLS–OLS discrepancy and its negative correlation with IV strength in observational studies, summarized in Figure 5 of Lal et al. (2024), which is reproduced in Section B in the Supplementary Materials.

The manual replication established benchmark 2SLS point estimates for each study and required substantial harmonization of heterogeneous replication materials. These coefficients serve as fixed ground truth for evaluating execution reliability on the original corpus. The AI-assisted workflow is first tested on its ability to reproduce the reanalysis of these 67 studies in Lal et al. (2024) from end to end. This includes extracting metadata from each paper, downloading replication materials, reconstructing the analyses, and generating standardized diagnostic reports using the authors' diagnostic template, which is implemented in the CRAN package `ivDiag` (Lal and Xu, 2024).

2.2. Forward Expansion and Within-Study Coverage

Beyond reproducing the original corpus, we extend the analysis in two dimensions. First, we incorporate 25 additional IV studies published in the same journals between 2023 and 2025 that satisfy the same inclusion criteria as the original corpus. The diagnostic template remains unchanged. This forward expansion allows us to assess whether the 2SLS–OLS discrepancy and its negative correlation with IV strength persist in recent work. Because these journals now condition acceptance on in-house verification—beginning with AJPS in 2015, followed by APSR and JOP around 2021—we expect high reproducibility in the expanded sample.

Second, rather than targeting a single specification per paper, the workflow replicates up to three IV specifications per study, including the baseline and key robustness variants. The unit of analysis is the IV specification. Expanding within-study coverage strengthens the execution test, as the workflow must identify and run distinct models within heterogeneous codebases without manual guidance.

TABLE 1. SAMPLE SIZE COMPARISON: ORIGINAL STUDY VS. THIS STUDY

	Lal et al. (2024)	**This Study**
Time Period Covered	2010–2022	2010–2025
Number of Studies	67	92 (67 original + 25 new)
Target Specifications per Study	1 baseline IV specification	Up to 3 IV specifications per study
Total Target Specifications	70 benchmark specifications	215 benchmark specifications
Manual Verification of Reproducibility	Yes	Yes (original 67); No (new 25)

Note: The unit of analysis in Lal et al. (2024) is IV designs (outcome-treatment-instrument combinations). Among the 67 papers, three contain two distinct IV designs, yielding 70 benchmark specifications. A specification means an outcome-treatment-instrument-covariate combination in a simple 2SLS regression.

Table 1 summarizes the expansion. Relative to Lal et al. (2024), the corpus grows both across studies (from 67 to 92) and within studies (from one benchmark specification per design to up to three per paper). The maximum number of evaluated specifications therefore increases from 70 to 215. This larger corpus provides a stronger test of whether the workflow can harmonize heterogeneous replication materials at scale.

2.3. Reproducibility Criterion and Evaluation

We adopt an intentionally minimal reproducibility criterion. For each IV specification, reproduction is deemed successful if the workflow exactly reproduces the reported 2SLS point estimate from the corresponding model using a harmonized system (which may differ from the one used by the original authors), namely,

$$\begin{aligned}\text{Coefficient in published work} &= \text{Coefficient generated by the authors' pipeline}\\ &= \text{Coefficient generated by the harmonized pipeline.}\end{aligned}$$

For the original 67-study corpus, exact agreement with the manually replicated 2SLS coefficient serves as the benchmark for reproducibility. In other words, for these 70 specifications, the equality between the published coefficient and the manually replicated coefficient (up to rounding error) has already been established. Our objective is therefore to verify the second equality: that the harmonized pipeline reproduces the same 2SLS estimate.

For the newly incorporated studies, reproducibility has not been manually verified. Although the journals from which these studies are drawn typically require exact numerical replication as a condition of publication, we do not independently verify the equality between the published and authors' pipeline outputs. Instead, we focus on achieving and documenting the second equality under our workflow. Verifying the first equality would sometimes require access to detailed Supplementary Materials, which are not always available, and is left for future work. Note that, as in Lal et al. (2024), reproducibility does not equal robustness or credibility, which requires additional diagnostics and knowledge about the research design.

Focusing on the 2SLS point estimate isolates the core execution bottleneck in large-scale reproducibility. Successful reproduction implies that the workflow has correctly identified the instrument set, endogenous regressor, control variables, fixed effects, sample restrictions, weights, and data transformations required for the model. Because the 2SLS coefficient is jointly determined by these components, exact numerical agreement indicates that the specification, data processing, and execution have been correctly harmonized. Conditional on this success, downstream diagnostic statistics are mechanically determined by the template. Reproducing the 2SLS point estimate therefore marks the critical breakpoint.

Having defined this criterion, we evaluate performance at both the paper and specification levels. We report ingestion success rates, specification extraction rates for up to three IV models per study, code execution rates, exact 2SLS replication rates at the specification level, and diagnostic report generation rates.

Because the pipeline is adaptive and version-controlled (see below), recurring execution failures are encoded as generalized rules between runs. As a result, conditional on accessible replication materials, stage-level success converges to full coverage under a fixed pipeline version. The metrics therefore assess whether the finalized executor can reliably harmonize heterogeneous replication packages and scale the 2SLS–OLS diagnostic analysis both forward in time and within studies.

3. AI-Assisted Reproducibility Workflow

This section describes the AI-assisted workflow used to execute the reanalyses of IV corpus defined in Section 2. The workflow targets a practical bottleneck in reproducibility: executing established research protocols reliably across heterogeneous replication packages while preserving numerical precision and auditability. This workflow does not automate methodological reasoning or introduce new statistical procedures. Instead, it standardizes and accelerates execution when usable data and code are available.

3.1. Design Principles

Large-scale reproducibility involves a basic tension between heterogeneity and determinacy. Replication materials vary widely across studies. Even within the IV corpus, papers differ in programming language (Stata, R, and Python), directory structure, naming conventions, and documentation quality. At the same time, reproducibility requires determinacy: for a fixed pipeline version and fixed inputs, numerical outputs must not depend on ad hoc decisions, platform-specific defaults, or stochastic behavior.

The workflow resolves this tension by separating adaptive coordination from fixed computation. Adaptation is used to route tasks, interpret failures, and select among predefined recovery steps. Numerical work—data preparation, model estimation, and diagnostic computation—is executed by version-controlled program code. For a fixed pipeline version and fixed inputs, the workflow produces identical numerical outputs and retains a complete audit trail of intermediate artifacts and logs. When new failure patterns are encountered, fixes are incorporated between runs and version-controlled, rather than allowing numerical behavior to drift within runs. Note that uncertainty estimates, particularly those based on bootstrap or jackknife procedures, make exact replication more challenging. This is because random seed behavior is not always portable across platforms, and parallel computation can further complicate seed control.

A second principle is the separation of scientific reasoning from execution. As described in Section 2, the diagnostic template—based on Lal et al. (2024) and `ivDiag`—is fixed in advance. Once this template is specified, reproduction reduces to execution-oriented tasks: acquiring replication packages, reconstructing computational environments, identifying IV specifications, running code, extracting estimates, and compiling diagnostic outputs. The workflow evaluates whether these tasks can be automated without loss of precision.

3.2. Architecture and Implementation

We implement the workflow using ***Claude Code Skills*** (hereafter, Skills), an agentic system organized as a three-layer architecture. At the top layer, an LLM (Claude) serves as an orchestrator that dispatches tasks, interprets errors, and determines how the pipeline proceeds. The middle layer consists of structured skill descriptions that define each stage's input–output contract and record previously resolved failure patterns. At the bottom layer, rule-based agent code and diagnostic scripts perform all file operations and statistical computation. The upper layers govern coordination and adaptation; the bottom layer governs numerical results.

The orchestrator reads task instructions, consults the relevant skill descriptions, writes and edits plain-text artifacts (primarily Markdown configuration files and small Python utilities), and invokes external tools as needed. It does not perform statistical estimation. All estimation and diagnostics are executed by explicit code in R, Stata, and Python. In particular, the full diagnostic suite is implemented in a standalone R script that operates solely on exported analysis datasets. The LLM therefore controls task routing and structured interpretation, but it is excluded from numerical estimation and inference. Figure 1 illustrate the three-layer architecture.

Reproduction is decomposed into modular stages because replication packages vary widely in software, directory structure, naming conventions, and documentation. Failures typically arise at distinct points—retrieval, parsing, code repair, execution, or reporting—and require stage-specific information. Each stage is assigned to a dedicated agent. Agents communicate exclusively through standardized intermediate files written to disk (e.g., JSON, CSV, logs) and share no hidden state. Each stage reads explicit inputs and writes explicit outputs, making every step inspectable and rerunnable. When a failure occurs, execution can resume from the affected stage without restarting the entire pipeline.

The system was developed iteratively through repeated encounters with diverse replication packages. When a new failure pattern is identified, the orchestrator proposes a diagnosis and candidate fix. After review, successful fixes are incorporated as version-controlled updates to the execution layer and implemented between runs rather than during execution. Each run is therefore tied to a fixed, inspectable pipeline version, and expanded coverage does not alter the numerical behavior of prior versions.

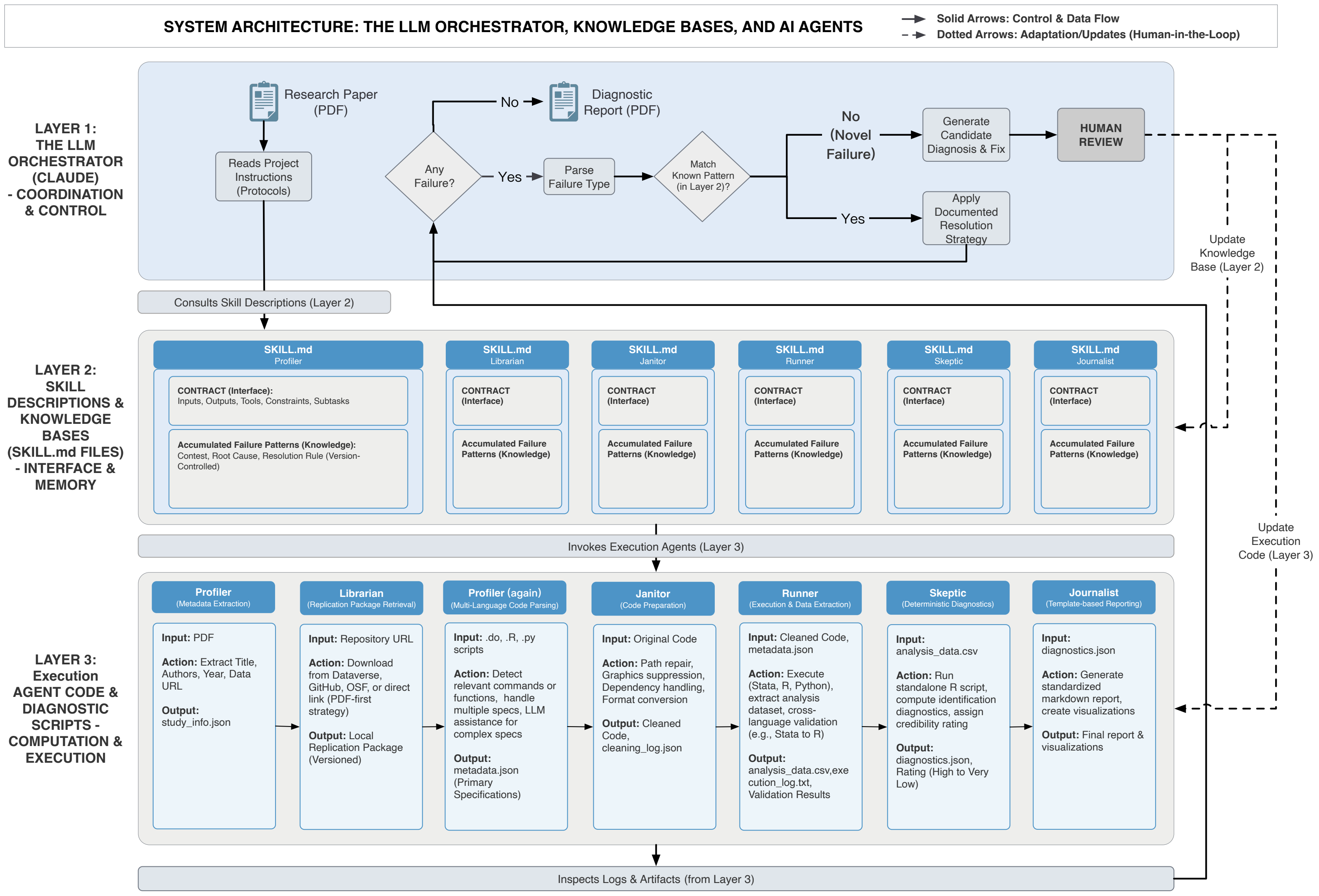

SYSTEM ARCHITECTURE: THE LLM ORCHESTRATOR, KNOWLEDGE BASES, AND AI AGENTS
Solid Arrows: Control & Data Flow
Dotted Arrows: Adaptation/Updates (Human-in-the-Loop)
LAYER 1: THE LLM ORCHESTRATOR (CLAUDE) - COORDINATION & CONTROL
Research Paper (PDF)
Reads Project Instructions (Protocols)
Any Failure?
No
Diagnostic Report (PDF)
Yes
Parse Failure Type
Match Known Pattern (in Layer 2)?
No (Novel Failure)
Generate Candidate Diagnosis & Fix
HUMAN REVIEW
Yes
Apply Documented Resolution Strategy
Update Knowledge Base (Layer 2)
Update Execution Code (Layer 3)
Consults Skill Descriptions (Layer 2)
LAYER 2: SKILL DESCRIPTIONS & KNOWLEDGE BASES (SKILL.md FILES) - INTERFACE & MEMORY
SKILL.md Profiler
CONTRACT (Interface): Inputs, Outputs, Tools, Constraints, Subtasks
Accumulated Failure Patterns (Knowledge): Contest, Root Cause, Resolution Rule (Version-Controlled)
SKILL.md Librarian
SKILL.md Janitor
SKILL.md Runner
SKILL.md Skeptic
SKILL.md Journalist
CONTRACT (Interface)
Accumulated Failure Patterns (Knowledge)
Invokes Execution Agents (Layer 3)
LAYER 3: Execution AGENT CODE & DIAGNOSTIC SCRIPTS - COMPUTATION & EXECUTION
Profiler (Metadata Extraction)
Input: PDF
Action: Extract Title, Authors, Year, Data URL
Output: study_info.json
Librarian (Replication Package Retrieval)
Input: Repository URL
Action: Download from Dataverse, GitHub, OSF, or direct link (PDF-first strategy)
Output: Local Replication Package (Versioned)
Profiler (again) (Multi-Language Code Parsing)
Input: .do, .R, .py scripts
Action: Detect relevant commands or functions, handle multiple specs, LLM assistance for complex specs
Output: metadata.json (Primary Specifications)
Janitor (Code Preparation)
Input: Original Code
Action: Path repair, Graphics suppression, Dependency handling, Format conversion
Output: Cleaned Code, cleaning_log.json
Runner (Execution & Data Extraction)
Input: Cleaned Code, metadata.json
Action: Execute (Stata, R, Python), extract analysis dataset, cross-language validation (e.g., Stata to R)
Output: analysis_data.csv,execution_log.txt, Validation Results
Skeptic (Deterministic Diagnostics)
Input: analysis_data.csv
Action: Run standalone R script, compute identification diagnostics, assign credibility rating
Output: diagnostics.json, Rating (High to Very Low)
Journalist (Template-based Reporting)
Input: diagnostics.json
Action: Generate standardized markdown report, create visualizations
Output: Final report & visualizations
Inspects Logs & Artifacts (from Layer 3)

Figure 1. Overview of the agentic AI workflow for reproducibility. The above figure illustrates the three-layer agentic architecture enabled by Skills. A top-layer LLM orchestrator routes tasks and interprets errors but does not perform estimation. The middle layer defines structured input–output contracts and records resolved failure patterns. The bottom layer consists of rule-based agent code and diagnostic scripts in R, Stata, and Python that execute all file and statistical operations through a modular seven-stage pipeline, from material acquisition to standardized reports.

The pipeline comprises seven stages, also illustrated in Figure 1. Given a paper in PDF format, an agent called **Profiler** extracts metadata and replication links. The **Librarian** downloads associated data and code. Then, the same **Profiler** agent identifies IV specifications within the replication code, including up to three targeted specifications per study as defined in Section 2. The **Janitor** prepares scripts for execution by resolving path dependencies and environment assumptions. The **Runner** executes the models and extracts 2SLS estimates. The **Skeptic** applies the fixed diagnostic template for IV designs. The **Journalist** compiles standardized reports. Each stage writes inputs and outputs to disk and records runtime, status, and error messages in structured logs, which form the basis for the stage-level performance metrics reported in Section 5.

3.3. Challenges in a Skills-Based Workflow

Using a Skills-based orchestration layer introduces practical challenges that are easy to understate if one only describes the workflow at a high level. We highlight four challenges that arise specifically because task routing and failure handling are AI-assisted, and we summarize how our implementation constrains these risks.

First, agent-based workflows can introduce run-to-run variation through branching logic. Even when numerical code is fixed, differences in routing or recovery steps can change which scripts are executed, which intermediate datasets are exported, or which specification is treated as primary. We constrain routing decisions to explicit, inspectable artifacts and fixed rules wherever possible. Agents define subsequent inputs through standardized files, and the orchestrator selects among documented options while recording all decisions in logs. For a fixed pipeline version and fixed inputs, reruns produce identical numerical outputs.

Second, coverage can expand in ways that are difficult to audit. Without discipline, a workflow may accumulate undocumented heuristics or paper-specific patches. We address this by version-controlling all deterministic code and recurring resolution rules. When a new failure pattern is identified, the fix is implemented as a generalized update between runs rather than as an ad hoc intervention within a single execution. Each run is therefore anchored to a stable pipeline state.

Third, execution variability across replication packages generates complex failure modes. Differences in software environments, directory structures, naming conventions, data encodings, and multi-language codebases create errors that cannot be fully anticipated ex ante. Some failures do not produce explicit error messages but instead appear as inconsistencies across intermediate artifacts, such as mismatched coefficients or incomplete exports. The workflow addresses these patterns through a structured resolution cycle. The orchestrator inspects logs and cross-validates outputs, traces failures across stages to identify root causes, and implements fixes as generalized rules in deterministic code. Resolved patterns are recorded in a structured knowledge base, allowing solutions to persist across runs while preserving numerical determinacy within each version.

Finally, supporting additional research designs creates pressure to embed design-specific assumptions in the executor. We avoid this by keeping the executor design-agnostic and treating diagnostic templates as inputs. Researchers specify estimands, estimators, and summary measures in the template. The executor performs the same execution tasks—acquisition, preparation, execution, extraction, and reporting—while invoking the appropriate template. Extending to new designs therefore requires modifying templates rather than rewriting the underlying architecture.

4. Demonstration

The previous section described each pipeline stage in abstract terms. Here we trace a single AJPS study, Rueda (2017), through the full workflow, from the input PDF to the final diagnostic report. At each stage, we show the intermediate artifacts the system generates, illustrating how the modular, file-based architecture supports auditability.

This demonstration reflects the cumulative knowledge embedded through the adaptive mechanism described above. To name a few nontrivial challenges the workflow has learned to handle: parsing the `ivreg2` syntax in the author's Stata code; resolving time-series operators such as `l.` and `l4.` embedded in variable names; enforcing the `e(sample)` post-estimation restriction; and executing a 13-script build chain of `.do` files. These capabilities were acquired from recurring patterns in earlier papers, indicating that the accumulated knowledge generalizes beyond the initial development set.

Rueda (2017) studies the relationship between polling station size and vote buying in Colombia, instrumenting average polling station size with the maximum size set by election authorities. This case is well suited for demonstration for three reasons. First, the design is straightforward, with one endogenous regressor and a clearly defined instrument. Second, the paper reports three IV specifications with strong first stages and adequate power, consistent with the diagnostics in our template. Third, the replication code is written in Stata, the dominant environment in our evaluation corpus and comparatively difficult for AI systems to parse and execute due to its closed-source ecosystem and idiosyncratic syntax. We now walk through the pipeline step by step.

Stages 1–2: Acquire Materials

The workflow begins with the paper's PDF as the only input. In this first stage, the Profiler parses the PDF and extracts structured metadata, including the title, author, journal, and, critically, the data repository URL embedded in the data availability statement. The output is a JSON file:

```
ivreg2 e_vote_buying l4.margin_index2 l.nbi_i ///
     l.own_resources lpopulation l.armed_actor ///
     l4.lsize lpotencial ///
     (lm_pob_mesa = lz_pob_mesa_f) if e(sample), ///
     first cluster(muni_code)
```

In Stage 2, the Librarian resolves the Dataverse DOI, queries the Dataverse API for the full file listing, and downloads the complete replication package. The materials include multiple Stata data files (`.dta`), dataset-construction scripts, analysis scripts, and a master analysis file. All files are stored in a versioned workspace directory. The download log confirms that both the article and the replication package were retrieved without error.

Stage 3: Determine Specifications

With the replication materials in place, the Profiler parses all Stata scripts to identify IV estimation commands. In the main analysis file (`main_results_aggregate_data.do`), it detects three `ivreg2` calls—a user-contributed two-stage least squares routine with syntax distinct from the built-in `ivregress 2sls`. Support for `ivreg2` was added incrementally as similar patterns were encountered in earlier papers. The first command appears as follows:

```
{
  "title": "Small Aggregates, Big Manipulation: Vote Buying Enforcement
   and Collective Monitoring",
  "authors": "Miguel R. Rueda",
  "year": "2017",
  "journal": "American Journal Of Political Science",
  "replication_url": "http://dx.doi.org/10.7910/DVN/K6ZOOW"
}
```

A central feature of this specification is the use of Stata time-series operators. For example, `l.nbi_i` denotes the first lag of `nbi_i`, and `l4.margin_index2` denotes the fourth lag. These operators must be mapped to realized variable names in the constructed dataset. The Profiler and Runner resolve this mapping using translation rules accumulated during prior executions.

For each `ivreg2` call, the Profiler constructs a structured representation that records the outcome, endogenous regressor, instrument, controls, sample restriction, and clustering variable. In this case, all three specifications share the same treatment and instrument, cluster at the municipality level, and differ only in the outcome variable or the set of controls. Rather than reproducing the full table of variables, we emphasize that the extracted structure captures only role assignments and estimation options necessary for downstream execution and diagnostics.

All extracted fields are written to `metadata.json`, which serves as the contract between the identification stage and subsequent execution and evaluation stages.

Stages 4–5: Code Preparation and Execution

The original replication package is organized as a multi-file Stata project. Dataset-construction scripts generate intermediate `.dta` files, which are subsequently loaded by analysis scripts to produce the reported tables. The IV specifications are embedded within this build chain rather than implemented in a single script. Reproducing the results therefore requires executing the scripts in the correct order and preserving dependencies across files.

Before execution, the workflow harmonizes the code for automated use. The Janitor redirects file paths to the local workspace and removes interactive or side-effect commands such as `graph export` and `log using`, which would otherwise interrupt batch execution. It also instruments each `ivreg2` call so that coefficients and standard errors can be programmatically extracted from the log. These adjustments do not alter the statistical content of the original code; they standardize the environment for reproducible execution. All modifications are recorded in structured `JSON` logs, preserving a complete audit trail of the harmonization process.

A substantive complication arises from the use of `if e(sample)` in the original Stata commands. This condition restricts the estimation sample to observations selected in a prior regression. Preserving the author's intended sample requires respecting this sequencing: the conditioning regression must be executed first, and data export must occur before subsequent commands overwrite the sample indicator. Correct handling of this dependency is essential for reproducing the reported specifications.

The Runner then executes the harmonized build chain in batch mode. All scripts complete without error. The system extracts the IV estimates from the marked log output and exports the corresponding analysis datasets for downstream diagnostics. As a cross-language validation step, each specification is re-estimated in R using `iv_robust()` from the `estimatr` package. The resulting coefficients match the Stata outputs within numerical tolerance, confirming that the harmonized execution preserves the original estimates.

Stage 6: Diagnostic Analysis

The Skeptic reads `metadata.json` to recover the model specification and loads the corresponding `analysis_data_spec_*.csv` files. After matching variable names to dataset columns, it invokes the R diagnostic script (`diagnostics_core.R`), which implements the full set of procedures described in Section B in the Supplementary Materials. Because the original IV designs use a single instrument, the script also applies the tF procedure (Lee et al., 2022), which adjusts critical values for the conventional t-test as a function of the first-stage F-statistic. Table 2 reports the results.

All three specifications display strong first stages. The effective F-statistics range from 204 to 8,598, well above the conventional cutoff of 10, and no model is flagged for weak instruments. Maximum polling station size is therefore a strong predictor of average station size in each specification.

For Specifications 1 and 3, the 2SLS estimates are statistically significant using the conventional t-test, bootstrap methods, tF test, and Anderson-Rubin test, and the jackknife ranges are tight. The estimates from the full and reduced control sets are similar in sign and magnitude, providing an internal robustness comparison within the paper's design.

For Specification 2, the instrument strength remains high, but inference is less robust. The p-value for the Anderson-Rubin test exceeds 0.05, the tF test does not reject, and the bootstrap-t interval includes zero. The jackknife identifies municipality `11001` as influential; removing it shifts the estimate by 58%. Under the prespecified rules, these two warnings yield a MODERATE rating. The system does not offer a substantive interpretation. It applies the same criteria to each specification and reports the results. Whether the weaker evidence reflects lower power in the smaller sample or a different relationship for the alternative

Table 2. Diagnostic results for Rueda (2017)

	Specificaiton 1	Specificaiton 2	Specificaiton 3
Outcome variable	`e_vote_buying`	`sum_vb`	`e_vote_buying`
Treatment variable	`lm_pob_mesa`	`lm_pob_mesa`	`lm_pob_mesa`
Instrument	`lz_pob_mesa_f`	`lz_pob_mesa_f`	`lz_pob_mesa_f`
Clustering varaible	`munni_code`	`munni_code`	`munni_code`
#Covariates	7	7	2
Instrument strength			
Effective F	827.2	203.9	8,598.3
Bootstrap F	925.5	202.6	8,989.5
2SLS estimate			
Coefficient	−1.460	−2.242	−0.984
Std. error	0.463	1.300	0.142
p-value for t-stat	0.002	0.085	0.000
#Observations	4,352	1,069	4,352
#Clusters	1,098	632	1,098
Robust inference			
AR p-value	0.002	0.075	0.000
tF $p < 0.05$	Yes	No	Yes
Bootstrap-c 95% CI	[−2.42, −0.62]	[−4.66, −0.07]	[−1.32, −0.73]
Bootstrap-t 95% CI	[−2.37, −0.55]	[−5.74, 1.26]	[−1.23, −0.73]
Any CI includes 0?	No	Yes: Boot-t	No
Sensitivity (jackknife)			
Range	[−1.49, −1.31]	[−2.27, −0.94]	[−0.99, −0.96]
Most influential	muni 11001 (Δ=0.15)	muni 11001 (Δ=1.30)	muni 11001 (Δ=0.02)
OLS comparison			
OLS coefficient	−0.626	−0.984	−0.675
2SLS/OLS ratio	2.3	2.3	1.5
Robustness rating	**HIGH**	**MODERATE**	**HIGH**

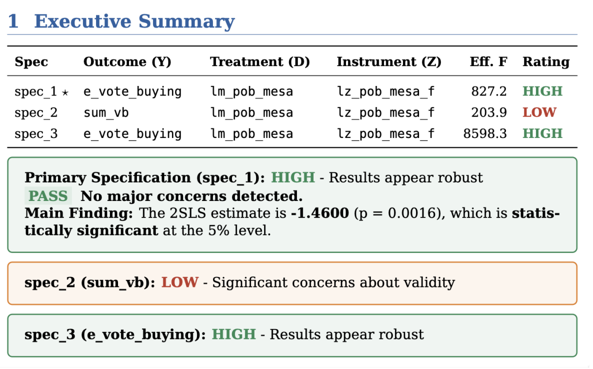
Notes: All bootstrap tests use 1,000 iterations with cluster-level resampling. Jackknife analysis removes one municipality cluster at a time. The effective F-statistic (Montiel Olea and Pflueger, 2013) is reported as the primary measure of instrument strength. Spec 2 triggers two warnings: (1) the Anderson–Rubin p-value exceeds 0.05; (2) removing municipality 11001 (Bogotá) changes the estimate by 58%, exceeding the 20% sensitivity threshold.

outcome remains a question for the researcher.

Note that the reanalysis does not assess the credibility of the core identification assumptions, namely unconfoundedness and the exclusion restriction of the instrument. When there is a single instrument for a single endogenous regressor, these assumptions are not directly testable, and the diagnostics cannot adjudicate their validity.

Stage 7: Report Findings

The Journalist then assembles the outputs from Stage 6 into a standardized report, `report.pdf`, which is included in the Supplementary Materials (Section E). The report is organized by specification. It begins with an executive summary table that lists the trustworthiness ratings. It then documents the study design, variable definitions, replicated IV estimates, and the full set of diagnostics. For each specification, four figures are generated automatically: following Lal et al. (2024), a coefficient comparison plot displaying OLS and 2SLS estimates with multiple confidence intervals, a comparison of first-stage F-statistics, bootstrap confidence intervals, and jackknife sensitivity. The format is identical across papers, which facilitates cross-study comparison.


1 Executive Summary

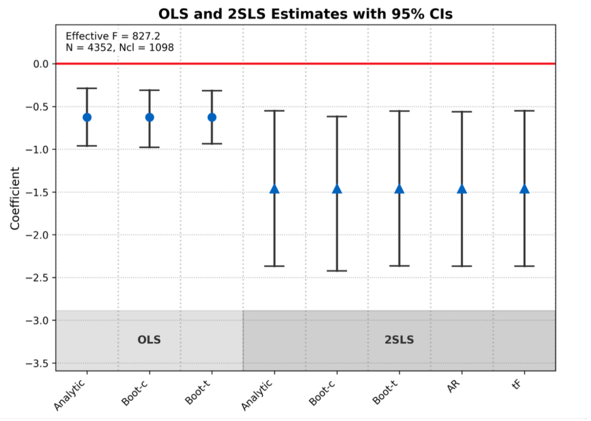
Spec	Outcome (Y)	Treatment (D)	Instrument (Z)	Eff. F	Rating
spec_1 ⋆	e_vote_buying	lm_pob_mesa	lz_pob_mesa_f	827.2	HIGH
spec_2	sum_vb	lm_pob_mesa	lz_pob_mesa_f	203.9	LOW
spec_3	e_vote_buying	lm_pob_mesa	lz_pob_mesa_f	8598.3	HIGH

Primary Specification (spec_1): HIGH - Results appear robust
PASS **No major concerns detected.**
Main Finding: The 2SLS estimate is **-1.4600** (p = 0.0016), which is **statistically significant** at the 5% level.

spec_2 (sum_vb): LOW - Significant concerns about validity

spec_3 (e_vote_buying): HIGH - Results appear robust

(a) Executive summary



(b) Coefficient plot for Spec 1

Figure 2. Selected figures from the diagnostic report for Rueda (2017) generated by the AI workflow. Left: the executive summary page summarizing ratings for all three specifications (two HIGH, one MODERATE). Right: the coefficient comparison plot for specification 1, showing OLS and 2SLS point estimates with analytic, bootstrap-c, bootstrap-t, tF, and Anderson-Rubin confidence intervals.

The full pipeline—from PDF ingestion and downloading the replication package to report generation—completed in less than four minutes. Diagnostic computation in Stage 6 accounts for most of the runtime. No human intervention was involved. Click [here] for a real-time demonstration, although the implementation itself does not rely on a graphical interface.

5. Main Findings

This section reports the main findings from implementing the AI-assisted replication workflow at scale. We first evaluate the workflow's performance, including its success rate, binding constraints, and execution time. We then present the empirical results from the extended IV corpus, showing that the automated pipeline reproduces the core patterns documented in Lal et al. (2024) under the same diagnostic template.

5.1. Performance of the AI Workflow

We evaluate the workflow on the 67 IV papers that form the benchmark corpus in Lal et al. (2024) and on 25 newly collected studies under identical inclusion criteria. Table 3 reports stage-level and end-to-end success rates at both the paper and specification levels.

For the original sample, end-to-end autonomous success is 55/67 (82%). All failures occur at the material retrieval stage, where replication archives are no longer publicly available. These 12 papers—all published before 2020—cannot currently be downloaded from public repositories. Using archived materials retained from the earlier project, however, we execute the pipeline and reproduce their findings. Conditional on accessible data and code, specification extraction, execution, and diagnostic analysis succeed for all papers. Thus, among studies with available materials, the current pipeline achieves full coverage.

Performance generalizes to the expanded sample. For the 25 newly incorporated papers, all stages succeed, yielding a 100% end-to-end rate. At the specification level, all 215 models are successfully extracted, executed, and analyzed; the 90% overall rate reflects out-of-scope designs rather than execution failures.

At first glance, this result may seem surprising. Three factors account for it. First, the system is version-controlled and adaptive: recurring execution issues are encoded as

Table 3. Success Replication Rate (Original and Expanded Samples)

A. *Original Sample* (67 papers)	Input	Success	Failure	Success rate
Material retrieval (Librarian)	67	55	12	82%
Specification extraction (Profiler)	67	67	0	100%
Code execution (Runner)	67	67	0	100%
Diagnostic analysis (Skeptic)	67	67	0	100%
End-to-end success rate				**55 / 67 = 82%**
Success rate given data*				**67 / 67 = 100%**

B. *Expanded Sample* (25 papers)	Input	Success	Failure	Success rate
Material retrieval (Librarian)	25	25	0	100%
Specification extraction (Profiler)	25	25	0	100%
Code execution (Runner)	25	25	0	100%
Diagnostic analysis (Skeptic)	25	25	0	100%
End-to-end success rate				**25 / 25 = 100%**

C. *All 215 Specifications*	Input	Success	Failure	Success rate
Specification extraction (Profiler)	215	215	0	100%
Code execution (Runner)	215	215	0	100%
Diagnostic analysis (Skeptic)	215	215	0	100%
End-to-end success rate				**215 / 215 = 100%**

Notes. Panels A and B report paper-level success: a paper succeeds at a given stage if at least one specification passes. Panel C reports specification-level success. "Failure" at the material retrieval stage reflects that, among the 67 original papers, replication materials for 12 are no longer publicly available online and were manually supplied from archived copies.

repair rules and therefore do not recur under a fixed pipeline version. Second, as noted above, the papers in the expanded sample were published after 2023, when all three journals required in-house replication. Third, the diagnostic template is deliberately narrow and well defined. Once the benchmark 2SLS point estimate is reproduced, downstream diagnostics are mechanically determined by the template. Conditional on correct specification parsing and estimation, the remaining steps proceed deterministically.

The workflow substantially reduces time and monetary cost relative to manual replication. For papers with accessible materials, end-to-end processing completes within minutes per paper, ranging from under one minute to half an hour, and can be fully parallelized. Most wall-clock time is spent downloading replication packages and running LLM diagnostics; only a small share of computation requires LLM calls. Once the code is stabilized in the execution layer, the marginal cost per additional paper is low. Researchers remain responsible for interpreting diagnostic reports, but core execution is automated.

5.2. Empirical Results from the Extended IV Corpus

We apply the automated workflow to the extended IV corpus using the same diagnostic template as in Lal et al. (2024). The main empirical patterns, after the AI workflow applies the diagnostic template, remain similar to those reported in the original study.

For example, the most important empirical finding in Lal et al. (2024) is the discrepancy between 2SLS and OLS estimates, and the negative relationship between the 2SLS–OLS ratio and first-stage strength, but only in observational studies. We find a similar pattern using estimates produced by the AI workflow, now with 215 specifications (versus 70 in the original study). Figure 3 mirrors Figure 5 in Lal et al. (2024). Subfigure (a) plots normalized coefficients and shows that 2SLS and OLS estimates generally share the same sign, with 2SLS magnitudes often larger. Subfigure (b) reports the distribution of the absolute ratio $|\hat{\tau}_{\text{2SLS}}/\hat{\tau}_{\text{OLS}}|$. In the extended sample, the mean of this ratio is 10.4 and the median is 3.0. In most of specifications, the 2SLS estimate exceeds the OLS estimate in absolute value. Subfigure (c) relate $|\hat{\tau}_{\text{2SLS}}/\hat{\tau}_{\text{OLS}}|$ to first-stage strength, measured by $|\hat{\rho}(d, \hat{d})|$. Among observational designs, regressing the log ratio on first-stage strength yields a robust negative correlation ($p = 0.000$, with the standard error clustered at the study level). Among experimental designs, the relationship is statistically indistinguishable from zero ($p = 0.391$). Subfigure (d) highlights observational studies in which the OLS estimates are statistically significant at the 5% level and are presented as part of the paper's main findings; the same pattern remains.

These findings reinforce the argument in Lal et al. (2024) that many estimates from observational IV designs rest on fragile identification assumptions, including instrument unconfoundedness and the exclusion restriction. In practice, such designs are often introduced to strengthen the credibility of causal claims based on "naive" OLS; however, when these assumptions fail, especially in the presence of publication bias toward statistically significant

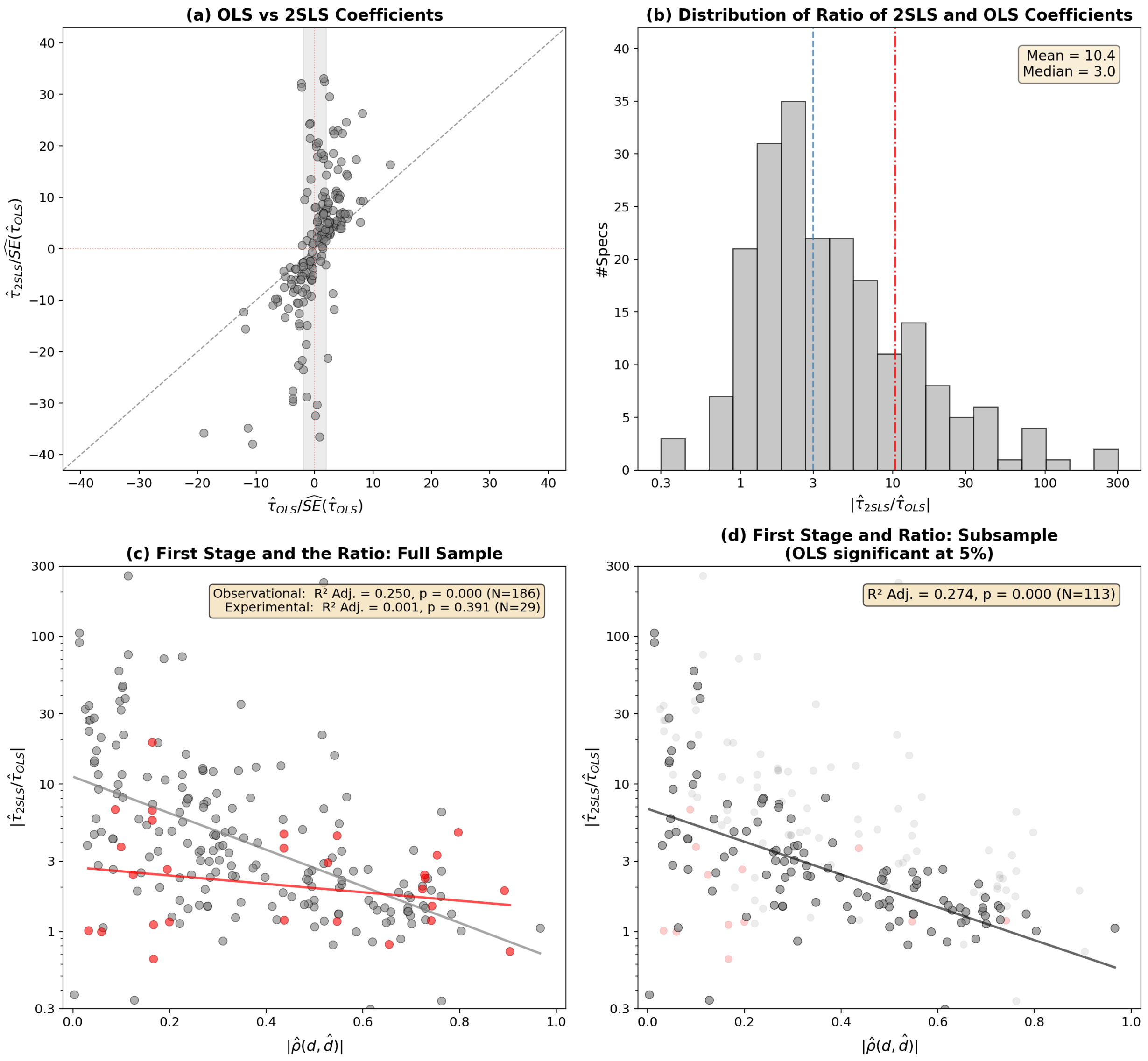


Figure 3. Relationship between OLS and 2SLS estimates. This figure replicates Figure 5 in Lal et al. (2024). Panel (a) rescales both coefficients by the reported OLS standard errors; the shaded region corresponds to the interval $[-1.96, 1.96]$. Panel (b) presents the distribution of the log absolute ratio between the reported 2SLS and OLS coefficients. Panels (c) and (d) examine how first-stage strength, measured by $|\hat{\rho}(d, \hat{d})|$, relates to the magnitude of the 2SLS-to-OLS ratio. Gray markers denote observational designs and red markers denote experiment-based instruments. Panel (d) further distinguishes designs in which the OLS estimate is statistically significant at the 5% level and is presented as part of the paper's primary results.

results, bias in 2SLS estimates may be substantially larger than in OLS.

A key difference, however, is the time required: whereas the original study involved approximately four years of manual data collection, processing, replication, and reanalysis, the expanded corpus is processed within days under the automated pipeline.

6. Discussion

This paper presents an AI-orchestrated workflow for systematic diagnostic evaluation of empirical research with publicly available data and code. The system executes end to end and generates standardized reports at scale with minimal human intervention. It relies, however, on diagnostic templates defined by human experts. While extraction and execution are automated, benchmark estimands, inclusion criteria, and warning thresholds reflect prior methodological judgment. The workflow therefore scales evaluation rather than defining evaluative standards. Related evidence from Straus and Hall (2026) shows that frontier coding models can reproduce and extend published political science analyses when given structured access to data and code.

Human-led replication initiatives remain indispensable. Journal-based data editor programs (e.g., the AEA Data Editor and *Political Analysis* replicators), research transparency organizations such as OSF and BITSS, the Institute for Replication (I4R), and independent replication communities continue to play a central role in credibility assessment. Evaluating identification strategies, measurement choices, data construction, and research design requires substantive judgment that cannot, at present, be delegated to automated systems. The workflow proposed here is complementary to these efforts, not a substitute.

We regard this separation between expert judgment and automated execution as a feature. Once templates and standards are specified, the pipeline applies them uniformly across studies, promoting consistency and transparency. As AI systems improve, elements of template design may become more adaptive. Because the architecture is modular, such changes can be incorporated without altering its core structure.

We outline extensions and discuss broader implications for empirical research.

6.1. Future Work

We consider three future extensions: expanding across research designs, deepening the scope of replication, and integrating the workflow into research infrastructure.

Extending to other designs. A natural extension applies the workflow to additional research designs for which we have previously constructed structured corpora. One candidate is panel studies estimated using estimators tied to two-way fixed effects (TWFE) models under variants of the parallel trends assumption. In earlier work, we assembled and reanalyzed a corpus of 49 such studies (Chiu et al., 2023). There, the benchmark object is the baseline TWFE estimate, and the diagnostic objective is to assess robustness to heterogeneous treatment effects using modern difference-in-differences estimators. Because the corpus was built under explicit inclusion criteria and a clearly defined estimand, it offers a disciplined template for automation. The workflow can replicate this corpus and extend it to newly published studies under the same criteria. A related extension concerns studies that estimate heterogeneous treatment effects using linear interaction models. Hainmueller, Mummolo and Xu (2019) compiled a corpus of 22 such applications. Together with the updated diagnostic template in Liu, Liu and Xu (2025), this corpus provides a structured foundation for scaling.

From diagnostics to full replication. The current workflow evaluates benchmark specifications. A natural next step is to reconstruct complete tables and figures and verify internal coherence across reported results. This would allow systematic comparison between reported and regenerated outputs. For example, the system could easily detect when two tables claim to use the same sample but report drastically different numbers of observations, or when a robustness specification silently changes clustering levels.

More broadly, the workflow makes previously costly diagnostics more feasible. It can automate cluster jackknife and leave-one-out procedures, compute leverage and Cook's distance, apply alternative winsorization thresholds, and implement permutation or bootstrap inference under alternative resampling schemes. In most empirical settings, these analyses are technically possible but rarely reported because they require substantial time and computation. By standardizing and parallelizing these procedures, the system lowers the cost of applying them across many studies. Scaling from individual specifications to integrated studies shifts attention from isolated coefficients to the stability of empirical claims as a whole.

Integration with the research process. The workflow can also integrate into the research and publication pipeline. It can assist authors in preparing replication materials through standardized, machine-readable structures that clarify file organization and specification definitions. Automated validation can identify missing dependencies, unresolved paths, or incomplete documentation at submission rather than after publication.

If journals adopt such tools, reproducibility checks could become more routine. Centralized verification services or third-party replication teams could rely on standardized diagnostic outputs, reducing the burden on editors and shortening review cycles. In this role, the workflow functions not only as a diagnostic instrument but as part of the infrastructure that supports more systematic verification.

6.2. Implications for Empirical Research

We expect this or similar workflows to influence how empirical research is conducted, evaluated, and accumulated in the coming years. We outline several implications below.

Lowering the cost of verification dramatically. The most immediate implication is a substantial reduction in the marginal cost of verification. When replication materials are available, recomputing estimates and applying standardized diagnostics becomes considerably less expensive than under current practice. This does not resolve disputes about identification or theory, but it changes incentives. At present, systematic verification is rare because its cost often exceeds its expected benefit. As that cost falls, more journals may find it feasible to require in-house or third-party reproducibility checks as a condition for acceptance. Authors, anticipating a higher likelihood of auditing, may adopt more disciplined coding practices and address influence, clustering, and resampling concerns ex ante. Verification thus becomes more closely integrated into the publication process rather than applied only after controversies arise.

Standardizing diagnostic reporting. Uniform diagnostic protocols can also reshape reporting norms. If weak-instrument tests, robust inference procedures, and sensitivity analyses are implemented automatically and summarized in standardized formats, discretion in how robustness is selected and presented declines. Journals may require structured diagnostic summaries alongside main results, much as data-availability statements have become routine. Referees may increasingly expect influence diagnostics and alternative clustering checks as part of the baseline empirical presentation. Over time, graduate training may adapt to treat such diagnostics as integral components of empirical analysis rather than supplementary exercises.

Enabling large-scale reanalysis and accelerating methodological research. Harmonized analysis datasets and structured metadata enable large-scale reanalysis under consistent criteria. Each of our previous large-scale reanalysis projects took three to four years of sustained effort. Much of that time was spent harmonizing replication materials, clarifying benchmark estimands, and standardizing robustness checks across heterogeneous applications. With the present workflow, many of these steps can be automated, substantially reducing the time required to conduct comparable large-scale reanalyses.

In the near term, this infrastructure may support more frequent and systematic reassessment of empirical literatures. Research groups, professional associations, or journals could periodically revisit published findings using updated diagnostic standards without incurring multi-year coordination costs. As harmonized corpora accumulate, empirical claims may compete not only on substantive grounds but also on demonstrated stability under shared diagnostics.

A large quantity of harmonized data will have profound implications for methodological research. In computer science, benchmark datasets such as ImageNet (Deng et al., 2009), MS COCO (Lin et al., 2014), SQuAD (Rajpurkar et al., 2016), and GLUE (Wang et al., 2018) structured progress by providing common evaluation environments. Researchers could compare algorithms under identical tasks and metrics, which facilitated cumulative improvement. Analogously, a large collection of harmonized empirical datasets with standardized diagnostic outputs can serve as a benchmark platform for causal and statistical methods. Methodologists could evaluate new estimators and inference procedures across diverse real-world applications rather than relying primarily on stylized simulations. By lowering the cost of empirical validation, the workflow may help shift methodological research toward cumulative comparison under shared empirical settings.

Taken together, these implications suggest that agentic AI workflows for reproducibility, with humans in the loop, can function as research infrastructure. They do not replace researchers' substantive judgment, but make systematic evaluation easier to conduct and harder to avoid. By lowering the cost of verification, standardizing diagnostics, and accelerating methodological development, they may help help make transparency and cumulative scrutiny part of routine empirical practice.

References

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.

Berge, Laurent. 2023. *fixest: Fast Fixed-Effects Estimation.* R package version X.X.X.
URL: *https://CRAN.R-project.org/package=fixest*

Blair, Graeme, Jasper Cooper, Alexander Coppock, Macartan Humphreys and Luke Sonnet. 2024. *estimatr: Fast Estimators for Design-Based Inference.* R package version 1.0.4.
URL: *https://CRAN.R-project.org/package=estimatr*

Chiu, Albert, Xingchen Lan, Ziyi Liu and Yiqing Xu. 2023. "Causal panel analysis under parallel trends: lessons from a large reanalysis study." *American Political Science Review* pp. 1–22.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE pp. 248–255.

Hainmueller, Jens, Jonathan Mummolo and Yiqing Xu. 2019. "How much should we trust estimates from multiplicative interaction models? Simple tools to improve empirical practice." *Political Analysis* 27(2):163–192.

Lal, Apoorva, Mackenzie Lockhart, Yiqing Xu and Ziwen Zu. 2024. "How much should we trust instrumental variable estimates in political science? Practical advice based on 67 replicated studies." *Political Analysis* 32(4):521–540.

Lal, Apoorva and Yiqing Xu. 2024. *ivDiag: Estimation and Diagnostic Tools for Instrumental Variables Designs.* R package version 1.0.6.
URL: *https://CRAN.R-project.org/package=ivDiag*

Leamer, Edward E. 1983. "Let's take the con out of econometrics." *The American Economic Review* 73(1):31–43.

Lee, David S., Justin McCrary, Marcelo J. Moreira and Jack Porter. 2022. "Valid t-Ratio Inference for IV." *American Economic Review* 112(10):3260–3290.

Lin, Tsung-Yi, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. In *European Conference on Computer Vision (ECCV).* Springer pp. 740–755.

Liu, Jiehan, Ziyi Liu and Yiqing Xu. 2025. "A Practical Guide to Estimating Conditional Marginal Effects: Modern Approaches." *arXiv preprint arXiv:2504.01355* .

Montiel Olea, José Luis and Carolin Pflueger. 2013. "A Robust Test for Weak Instruments." *Journal of Business & Economic Statistics* 31(3):358–369.

Open Science Collaboration. 2015. "Estimating the reproducibility of psychological science." *Science* 349(6251):aac4716.

Rajpurkar, Pranav, Jian Zhang, Konstantin Lopyrev and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. ACL pp. 2383–2392.

Rueda, Miguel R. 2017. "Small Aggregates, Big Manipulation: Vote Buying Enforcement and Collective Monitoring." *American Journal of Political Science* 61(1):163–177.
URL: *https://doi.org/10.1111/ajps.12260*

Straus, Graham and Andrew B. Hall. 2026. "How Accurately Did Claude Code Replicate and Extend a Published Political Science Paper?" Unpublished manuscript, January 9.
URL: *https://www.andrewbenjaminhall.com/Straus_Hall_Claude_Audit.pdf*

Torreblanca, Carolina, William Dinneen, Guy Grossman and Yiqing Xu. 2026. "The Credibility Revolution in Political Science.".
URL: *https://arxiv.org/abs/2601.11542*

Wang, Alex, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy and Samuel R. Bowman. 2018. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP*. ACL pp. 353–355.

Supplementary Materials

Scaling Reproducibility: An AI-Assisted Workflow for Large-Scale Reanalysis

A. System Architecture and Workflow
- **A.1. Three-Layer Architecture**
- **A.2. Agents and Stage-Level Implementation**

B. IV Diagnostic Template

C. Adaptive Execution Mechanism
- **C.1. The Resolution Cycle**
- **C.2. Adaptation in Practice**

D. Empirical Inventory and Performance
- **D.1. Classes of Implementation Variation**
- **D.2. Pipeline Performance**
- **D.3. Detailed Inventory of Resolved Issues**

A. System Architecture and Workflow

This section describes the three-layer system architecture and details of the AI workflow.

A.1. Three-Layer Architecture

The AI workflow described in the main text, which is adaptive in orchestration and deterministic in computation, is implemented through a three-layer system. The layers are ordered by control flow: the LLM orchestrator governs coordination, skill descriptions mediate task specification and accumulated knowledge, and deterministic agent code executes all operations whose outputs must be numerically reproducible.

Layer 1: The LLM Orchestrator. The orchestrator is an LLM (Claude) that manages the pipeline lifecycle. It reads a project-level instruction file that specifies global protocols for stage ordering, error handling, logging, and knowledge updates. For each stage, it consults the relevant skill description, prepares the required inputs, invokes the corresponding agent, and inspects the resulting logs and artifacts.

When a stage fails, the orchestrator parses structured log output to determine the failure type. If the error matches a previously recorded pattern in the relevant knowledge base, it applies the documented resolution strategy by modifying inputs or dispatching auxiliary steps. If the failure is novel, it generates a candidate diagnosis and proposed fix, which is subject to human review before being incorporated into the system. The orchestrator does not perform statistical estimation, transform datasets, or modify numerical routines. Its role is strictly coordinative: it decides *which* component runs and *how* to respond to execution outcomes, but it never enters the computational path that determines numerical results.

Layer 2: Skill descriptions and knowledge bases. Each agent is associated with a structured natural-language file (`SKILL.md`) that functions as both a formal interface specification and a persistent knowledge base.

The first component of the file defines the agent's contract: required inputs, expected outputs, permissible tools, execution constraints, and the sequence of subtasks. This specification ensures that the orchestrator interacts with the agent in a controlled and predictable manner.

The second component records accumulated failure patterns encountered during development and evaluation. Each entry documents the context in which a failure occurred, the root cause, and the generalized resolution rule. These entries are written in structured form to promote consistency across updates. When a new class of failure is resolved, the corresponding rule is added to the relevant skill file. This process expands the system's coverage without modifying deterministic computation within a given pipeline version. Because skill files are version-controlled, each run is associated with a fixed and inspectable knowledge state.

Layer 3: Deterministic agent code and diagnostic scripts. The bottom layer consists of deterministic program code that executes all file operations and statistical procedures. Each agent is implemented as an independent Python class responsible for a specific stage of the pipeline (e.g., metadata extraction, repository retrieval, specification parsing, code preparation, execution, or reporting). Agents operate only on explicit inputs and produce explicit outputs written to disk. They share no internal state.

Statistical estimation and diagnostic procedures are executed by explicit scripts in `R`, `Stata`, and `Python`. In particular, the full diagnostic suite is implemented in a standalone R script (`diagnostics_core.R`) that consumes exported analysis datasets and produces structured diagnostic outputs. This script calls established statistical packages, including `estimatr` (Blair et al., 2024), `fixest` (Berge, 2023), `boot`, and `ivDiag` (Lal and Xu, 2024). Given the same inputs and the same pipeline version, this layer produces identical numerical results across runs.

Information flows downward as instructions and dispatch decisions (Layer 1 to Layer 3) and upward as logs, intermediate artifacts, and error messages (Layer 3 to Layer 1). Adaptation occurs only through controlled updates to Layer 2 (skill descriptions) and, when necessary, to deterministic agent code between versions. Within a fixed version, numerical outputs depend exclusively on deterministic code and exported datasets.

This separation allows the system to remain adaptive in coverage while preserving computational determinacy in each execution. Human oversight operates at the boundary between Layers 1 and 2: proposed updates to knowledge bases or deterministic routines are reviewed before being committed. The result is an architecture that evolves across versions yet remains reproducible within version.

A.2. Agents and Stage-Level Implementation

This section describes the AI agents that implement each stage of the workflow. The focus is operational: inputs, outputs, implementation details, and design decisions that shaped the system through repeated encounters with real replication packages. Statistical diagnostics are described separately in Section B.

A.2.1. Data Acquisition: Profiler and Librarian

The first step in reproducing an empirical study is obtaining the paper itself and its accompanying replication package—typically comprising data files and analysis code. These tasks are handled by the Profiler and the Librarian. Together they illustrate the adaptive-orchestration-with-deterministic-computation principle at the pipeline's entry point: the Profiler interprets unstructured PDF text, while the Librarian executes deterministic retrieval routines.

Profiler (Metadata Extraction) Given a paper in PDF format, Profiler converts the PDF to plain text using `pdftotext`, then extracts structured metadata including title, authors, publication year, and journal. More critically, it searches for a "Data Availability Statement" (or equivalent section) and extracts the data repository URL—for example, links to Harvard Dataverse, GitHub, or the Open Science Framework (OSF).

Extraction combines deterministic parsing rules (e.g., URL detection, section matching) with limited language-model assistance when formatting is irregular or nonstandard. The extracted information is written to a standardized JSON file (`study_info.json`), which serves as the input to the Librarian.

Librarian (Replication Package Retrieval) The Librarian reads the repository URL recorded in `study_info.json` and downloads the complete replication package. It supports commonly used academic data hosting platforms:

- **Dataverse**: retrieves the dataset file list via the Dataverse API and downloads each file individually.
- **GitHub**: downloads the repository as a ZIP archive.
- **OSF**: retrieves project files via the OSF API.
- **Direct HTTP links**: downloads files via standard HTTP requests.

A key design choice is a PDF-first retrieval strategy: URLs embedded in the paper are prioritized over keyword-based search. Title-based search frequently returns irrelevant datasets for papers with common keywords. By contrast, URLs in the PDF typically point directly to the authors' own repository. Keyword search is used only as a fallback when no URL is found. If automated retrieval fails entirely, the user may manually supply the replication package so that downstream stages can proceed.

The output is a versioned local copy of the full replication package.

A.2.2. Specification Identification: Profiler (Again)

With the replication package available, Profiler (the same agent in charge of metadata extraction) identifies the 2SLS specifications implemented in the code. This is one of the most technically challenging stages of the pipeline.

Multi-Language Code Parsing The Profiler parses all Stata (`.do`), R (`.R`), and Python (`.py`) scripts in the replication package. It uses language-specific deterministic patterns to detect IV estimation commands.

For Stata, matched commands include `ivreg2`, `ivregress 2sls`, `reghdfe ... (D = Z)`, and `ivprobit`. For R, it detects `ivreg()`, `iv_robust()`, and `feols()` calls containing IV syntax. For Python, it detects `IV2SLS()` commands.

For each identified IV command, the Profiler extracts a structured representation of the specification, including the outcome variable (Y), endogenous treatment (D), instrument(s) (Z), exogenous controls (X), clustering variable, fixed effects, statistical software used, source script file, and, when available, the corresponding table reference in the paper. These fields are recorded in `metadata.json`, which is consumed by the Runner and Skeptic.

Multiple Specification Handling Empirical papers often report multiple IV specifications. The Profiler extracts *all* IV commands and then selects primary specifications through a multi-step procedure:

1. Deduplicate specifications based on (Y, D, Z, X) combinations.
2. Rank candidates using heuristic rules (e.g., explicit table references, location in main-results scripts, richer control sets).
3. Retain up to three primary specifications per study.

Language Model Assistance When deterministic parsing fails due to opaque variable names, sparse comments, or nonstandard syntax, the Profiler selectively invokes a lightweight language model to interpret the command structure. For example, given a complex Stata command, the model may identify which variable serves as the endogenous regressor and which as the instrument.

Patterns that initially require LLM interpretation are progressively encoded as deterministic parsing rules. Over time, this reduces reliance on model interpretation and increases reproducibility across standard command formats.

A.2.3. Execution Environment: Janitor and Runner

Janitor (Code Preparation) Replication packages are typically written under assumptions about directory structure and software environment. Direct execution in a different environment often fails due to path mismatches, unavailable graphics devices, deprecated packages, or syntax conventions.

The Janitor performs automated repairs to enable execution in the current environment. These operations include:

- **Path repair**: Replacing absolute paths (e.g., `C:\Users\author\data`) with workspace-relative paths. For Stata, this includes parsing variations in `use` syntax and macro references.
- **Graphics suppression**: Commenting out graphical commands (e.g., `graph export`, `pdf()`, `png()`) that may fail in headless environments.
- **Dependency handling**: Substituting or commenting out deprecated R packages (e.g., `rgdal`, `rgeos`, `maptools`).
- **Stata-specific repairs**: Handling `#delimit` syntax, macro substitution (e.g., `$datadir`), and panel declarations.
- **Data format conversion**: Converting nonstandard formats (e.g., `.tab`) to formats readable by Stata or R.

Each operation was added in response to a real failure class encountered during development. All modifications are recorded in a structured cleaning log (`cleaning_log.json`).

Runner (Execution and Data Extraction) The Runner executes the cleaned replication code in batch mode using the appropriate interpreter: `stata -b do`, `Rscript`, or `python3`, depending on the software type recorded in `metadata.json`.

The primary objective is to extract the analysis dataset containing all variables used in the reported specification. Data export commands are inserted at appropriate execution points, and the in-memory dataset is saved as CSV. For Stata studies, `.dta` files are also converted to CSV after execution.

For Stata-based studies, the Runner performs cross-language validation. It translates the Stata IV command into an equivalent R specification and re-estimates the model on the same dataset. Coefficients are compared within a tolerance of $\max(0.01 \times |\hat{\beta}|, 10^{-6})$. Translation must handle syntactic differences, such as converting `i.var` to `factor(var)` and mapping `absorb()` to fixed-effects syntax in `fixest`. The catalog of translation mappings expanded iteratively as new Stata idioms were encountered.

Outputs include the extracted dataset (`analysis_data.csv`), execution logs (`execution_log.txt`), and cross-language validation results.

A.2.4. Diagnostic Execution: Skeptic

The Skeptic applies the human-designed diagnostic template to the extracted dataset by invoking a standalone R script that implements the IV diagnostics described in Section B. The Skeptic contains no language-model calls and no adaptive logic. All statistical procedures are deterministic and version-controlled.

Diagnostic results, including all statistics and warning indicators, are written to a file called `diagnostics.json`, which serves as the input to the reporting stage.

After computing all diagnostics, the Skeptic assigns a summary credibility rating based on the number of triggered warning flags:

Table S1. Credibility Rating Scheme

Rating	**Condition**	**Interpretation**
High	No warnings	Strong instruments and robust inference
Moderate	1–2 warnings	Some concerns; interpret with caution
Low	3–4 warnings	Substantial validity concerns
Very Low	≥ 5 warnings	Results likely unreliable

This rating is an auxiliary summary indicator rather than a substitute for independent evaluation. Individual diagnostics may differ in substantive importance—for example, extremely weak instruments may be more consequential than several minor warnings. The rating is intended to facilitate screening and to highlight specifications that merit closer scrutiny.

A.2.5. Reporting: Journalist

The Journalist converts `diagnostics.json` into a standardized Markdown report and visualizations. The report includes model specification details, diagnostic statistics, and warning summaries. Standardized charts include first-stage F-statistics, bootstrap confidence intervals, and jackknife sensitivity distributions.

Because reporting is fully deterministic and template-based, identical inputs produce identical outputs, preserving cross-study comparability.

B. IV Diagnostic Template

This section briefly summarizes the statistical diagnostics applied to each IV specification. The full motivation and interpretation are detailed in Lal et al. (2024). The pipeline implements the same diagnostic template.

Instrument strength. Instrument strength is assessed using first-stage F-statistics. The effective F-statistic (Montiel Olea and Pflueger, 2013) serves as the primary indicator. Following common practice, $F < 10$ triggers a weak-instrument warning.

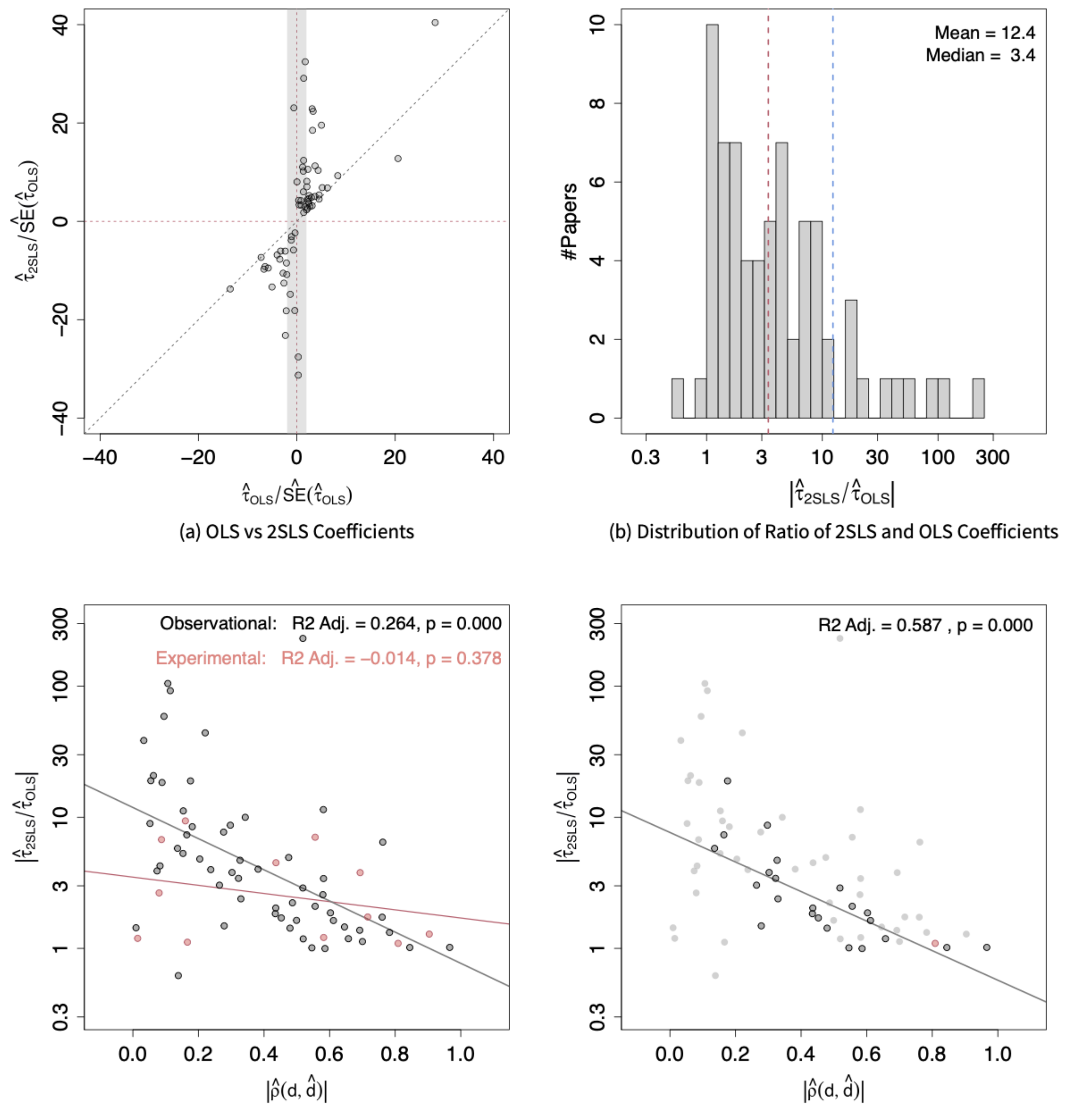
Robust inference. Inference robustness is evaluated using:

- The Anderson–Rubin (AR) test, which remains valid under weak instruments.
- Bootstrap confidence intervals (including cluster bootstrap when clustering is used).
- The tF procedure (Lee et al., 2022) in single-instrument cases, which adjusts critical values as a function of the first-stage F.

Sensitivity analysis. A leave-one-out jackknife procedure assesses the influence of individual observations or clusters on the IV estimate. Large changes relative to the baseline estimate trigger warnings.

2SLS–OLS comparison. An OLS model (without instruments) with the same outcome-treatment-covariates specification is estimated and compared to the 2SLS estimate.

Original findings. For comparison purposes, we reproduce Figure 5 of Lal et al. (2024), which is open access, below.



(a) OLS vs 2SLS Coefficients

(b) Distribution of Ratio of 2SLS and OLS Coefficients

(c) First Stage and the Ratio: Full Sample

(d) First Stage and Ratio: Subsample

Figure 5. Relationship between OLS and 2SLS estimates. In subfigure (a), both axes are normalized by reported OLS SE estimates with the gray band representing the $[-1.96, 1.96]$ interval. Subfigure (b) displays a histogram of the logarithmic magnitudes of the ratio between reported 2SLS and OLS coefficients. Subfigures (c) and (d) plot the relationship between $|\hat{\rho}(d, \hat{d})|$ and the ratio of 2SLS and OLS estimates. Gray and red circles represent observational and experimental studies, respectively. Subfigure (d) highlights studies with statistically significant OLS results at the 5% level, claimed as part of the main findings.

C. Adaptive Execution Mechanism

The preceding sections described the self-improvement mechanism in architectural terms. This section makes that mechanism concrete. We document the recurring resolution cycle through which new classes of implementation variation were addressed, and the role of structured knowledge accumulation in preventing recurrence. Representative examples are drawn from the 92-paper evaluation corpus. A complete empirical inventory of resolved issue classes appears in Appendix D.

C.1. The Resolution Cycle

The workflow expands coverage through a recurring four-phase cycle.

First, the system detects anomalies. These include execution failures, silent data inconsistencies, and mismatches between expected and observed outputs. It identifies them by reading execution logs, comparing estimates across software, and checking intermediate files for internal consistency.

Second, it identifies the root cause. Rather than retrying commands, it traces the failure across stages and programming languages to locate the source of the problem.

Third, it implements a general fix. The repair is written as a rule in the relevant deterministic code layer, not as a patch for a single paper. Future papers exhibiting the same pattern are handled automatically.

Fourth, it records the resolution. The context, problem, fix, and impact are documented in the agent's knowledge base (`SKILL.md`) in a standardized format so that the solution can be retrieved and reused.

This cycle differs from manual replication and fixed scripts. Manual replication repeats these steps for each paper, and lessons are rarely formalized. A fixed script applies predefined rules but cannot detect new failure classes or extend its rule set. Here, the same four steps are applied systematically, with human review limited to approving code updates before they are committed.

Adaptation proceeds through two channels. When a failure class is well-defined and recurring, the fix is encoded as a deterministic rule in the relevant agent's code. When a pattern is better captured as contextual guidance, it is recorded in the agent's knowledge base and used to guide future orchestration. In both cases, updates occur between runs and are version-controlled.

C.2. Adaptation in Practice

We describe every steps of the adaptation cycle below and illustrate them with examples from practice.

C.2.1. Discovery

Many consequential failures produced no explicit error message. Detection therefore relied on cross-field consistency checks.

Example: Silent coefficient failure due to backtick quoting. When CSV column names begin with an underscore (e.g., `_log_providers`), R internally wraps them in backticks. A call such as `coef(fit)["_log_providers"]` returns `NA` silently. The orchestrator detected this anomaly because first-stage statistics were computed correctly while downstream estimates—2SLS coefficients, jackknife results, bootstrap intervals—were missing. This pattern suggested a name-matching failure rather than a data or model error. The fix introduced a four-way lookup function that attempts raw, backtick-quoted, stripped, and fuzzy matches, and was applied across the diagnostic script.

Example: Destructive safeguard in Stata. The pattern `capture drop X; gen X = expr` prevents error `r(110)` when a variable already exists. However, if `expr` references a missing variable, `gen` fails after `drop` has removed the original variable. The orchestrator identified this when a replication package contained a precomputed variable that was inadvertently deleted. The fix replaced drop-and-recreate with a backup–restore pattern: rename the original variable, attempt generation, and restore if generation fails.

C.2.2. Root-Cause Diagnosis

Complex failures often spanned multiple agents and programming environments.

Example: Cross-language sign reversal. In one specification, R cross-validation yielded +2.15 while Stata produced −2.24. The orchestrator traced the discrepancy across seven layers: a dropped lagged control, variable-name abbreviation in Stata, incomplete panel export, and interaction with `e(sample)` memory. Switching to a full-panel export with an estimation-sample indicator resolved the discrepancy, producing an exact coefficient match (−2.2420). Diagnosis required coordinated reasoning across Python, Stata, and R components.

C.2.3. Generalized Code Repair

Fixes were implemented as general rules rather than paper-specific patches.

First, when a single failure revealed a broader pattern, we generalized the rule. For example, failure to recognize abbreviated `#delimit` commands led to expansion of the regular-expression coverage to include all valid abbreviations. The change applied to all subsequent papers using any shortened form.

Second, when repeated failures exposed a structural weakness, we redesigned the component. Retrieval based on keyword search frequently downloaded incorrect datasets. We therefore adopted a PDF-first retrieval strategy that prioritizes repository links embedded in the paper.

Third, when insertion errors arose during data export, we refined the code-injection logic. Exporting analysis data required detecting the active delimiter mode, scanning multi-line commands before insertion, and using content-based anchors rather than line numbers to avoid drift as earlier edits shifted subsequent lines.

C.2.4. Knowledge Accumulation

Each resolved issue is recorded in structured form:

`[Date] capture drop + gen safeguard destroys variables`

Context: Pattern `capture drop X; gen X = EXPR`.

Problem: `gen` failure deletes preexisting variable.

Fix: Backup–restore strategy.

Impact: Generalizable to all similar exports.

Entries are standardized, retrievable, and version-controlled. Across agents, the knowledge base expanded to 64 entries, covering syntax handling, name resolution, translation mappings, specification extraction, and workflow coordination.

D. Empirical Inventory and Performance

This section documents the full set of implementation-level issues encountered and resolved during development and evaluation of the AI-assisted reproduction pipeline across 92 IV studies. Each entry records a distinct execution pattern, the agent(s) adjusted, and the corresponding resolution. The detailed inventory is organized into ten classes. Together, they provide a concrete account of the recurring irregularities that arise when replication materials are executed in a standardized, automated environment.

D.1. Classes of Implementation Variation

Across the 92 studies, we identified and resolved failures spanning ten broad classes. Table S2 summarizes these classes, affected stages, and typical issues of failure patterns.

Table S2. Classes of variation encountered and resolved across 92 papers

Class	**Stage(s)**	**Representative pattern**
Path and environment	Janitor, Runner	Absolute paths, global macros, sub-file mismatches
Software and language	Profiler, Runner, Skeptic	Multi-language codebases, operator translation
Data format and encoding	Janitor, Runner, Skeptic	`.tab` ambiguity, factor encoding, quoting rules
Variable name mismatch	Skeptic, Profiler	Typos, prefix matches, unresolved macros
Stata syntax dialect	Janitor	`#delimit` modes, merge syntax, `e(sample)` handling
Model specification structure	Skeptic, Profiler	Panel FE, subset conditions, bandwidth rules
Runtime resource constraints	Skeptic	Memory limits, jackknife timeouts
Graphics and interactivity	Janitor	Graphics devices, output table packages
Data acquisition	Librarian	Deprecated links, incorrect datasets
Code injection logic	Janitor	Delimiter mode, line-number drift

These classes span the full pipeline: data acquisition, code parsing, environment preparation, execution, cross-language validation, and diagnostic computation. Some patterns reflect software dialect differences (e.g., Stata version changes); others reflect repository-specific formats or naming conventions. Many failures were silent and detectable only through cross-stage consistency checks.

The detailed inventory that follows lists each resolved issue, its manifestation, the deterministic repair implemented, and the responsible agent. The purpose is transparency rather than exhaustiveness of narrative: readers can trace each capability to a concrete failure pattern and code modification.

D.2. Pipeline Performance

Performance gains are substantial and empirically verifiable. The end-to-end success rate increased from approximately 63% in the initial implementation to 92.5% in the current version. These improvements were developed and evaluated on the same benchmark corpus; performance on previously unseen corpora may differ. Nevertheless, the resolved failure modes primarily reflect language-level and software-level conventions rather than idiosyncratic features of individual papers, suggesting that a meaningful portion of the gains should generalize.

The single largest improvement followed the adoption of a "PDF-first retrieval architecture," in which the system first analyzes the published PDF to infer the likely location of the replication package before attempting data acquisition. This design substantially reduced incorrect dataset downloads and increased the success rate from 62.9% to 88.6%.

Approximately 40% of resolved issue classes were first encountered during evaluation rather than anticipated ex ante. This pattern illustrates the limits of purely rule-based design. A fixed script can implement predefined rules, but it cannot account for patterns that were not foreseen. Here, new patterns were incorporated into the deterministic code base after diagnosis and review, expanding coverage across subsequent studies.

D.3. Detailed Inventory of Resolved Issues

Below we present an class-by-class inventory of resolved implementation issues.

Class 1: Path and environment variability. Replication code is typically written for a specific directory structure and operating system. When executed in a standardized workspace, these assumptions often fail. The patterns below record the path and environment issues encountered and the corresponding repairs.

Pattern	**Manifestation**	**Resolution**	**Agent**
Absolute paths	`cd "C:\Users\john\..."`, `setwd("/home/author/...")`	Regex detection; replace with relative paths or comment out	Janitor
Global path macros	Stata `global datadir "C:\..."` followed by `$datadir` references	Inline macro substitution before commenting out the `global` definition	Janitor

Pattern	**Manifestation**	**Resolution**	**Agent**
`use` command variants	Quoted paths, extensionless filenames, digit-prefixed names, macro-embedded paths	Multiple regex patterns covering all observed variants	Janitor
Sub-file reference mismatches	`do script.do` when actual filename is `script_rep.do`	Fuzzy stem-matching: search for candidates when exact match fails	Janitor
Subdirectory output files	`analysis_data.csv` generated in a subdirectory, not the expected root	Recursive `rglob()` fallback search	Runner
Platform-specific software paths	macOS vs. Linux Stata installation paths	Platform-aware path detection	Runner

Class 2: Software and language variability. The papers in the corpus use Stata, R, and Python, and several combine multiple languages within a single replication package. Supporting cross-language execution required systematic expansion of parsing and translation rules. The table below lists the software- and language-related issues resolved during development.

Pattern	**Manifestation**	**Resolution**	**Agent**
Multi-language codebases	Stata data preparation + R analysis in a single replication package	Profiler parses `.do`, `.R`, and `.py` files simultaneously	Profiler
Diverse IV commands	`ivreg2`, `ivregress`, `reghdfe`, `ivprobit`, `ivtobit`, `rdrobust fuzzy`, manual 2SLS	Expanding IV command pattern list; detecting manual two-stage implementations	Profiler
Stata $\rightarrow$ R variable translation	`l4.margin_index2` becomes `l4margin_index2` or `l4_margin_index2` in CSV	Three-tier resolution: exact $\rightarrow$ dot-stripped $\rightarrow$ underscore-separated $\rightarrow$ recompute from base	Runner, Skeptic, R core
Factor variable expansion	`i.year` $\rightarrow$ dummy columns `_Iyear_2000`, `_Iyear_2001`, etc.	Detect `i.` prefix; match expanded dummy patterns	Runner, Skeptic
Time-series operators	`L.`, `L4.`, `F2.`, `D.`, compound `L2D.var`	Dual-side parsing chains in Python (Runner/Skeptic) and R (`diagnostics_core.R`)	Runner, Skeptic, R core

Pattern	**Manifestation**	**Resolution**	**Agent**
Truncated variable names	Stata truncates to 32 characters with `~` (e.g., `incumbvotesmajor~t`)	Prefix-plus-tilde pattern matching	Runner
R formula objects with `update()`	Base formula `f <- y ~ x` modified by `update(f, . ~ . + z)`	Extract formula dictionary; apply `update()` rules to expand	Profiler
R interaction shorthand	`A * B` implying `A + B + A:B`	Detect `*` patterns; add explicit `A:B` to control list	Profiler

Class 3: Data format and encoding variability. Replication materials are distributed in multiple data formats and encoding conventions. Differences in file types, string encodings, and export formats required explicit handling to ensure consistent downstream computation. The following entries summarize the issues encountered.

Pattern	**Manifestation**	**Resolution**	**Agent**
`.tab` format ambiguity	Dataverse stores as `.tab` (TSV); Stata expects `.dta` (binary)	Detect `.tab` files; convert to CSV or rewrite `use` commands	Janitor
Factor-encoded strings	Column stored as `"0: Not low-education"` instead of numeric 0/1	Python preprocessing to convert to numeric before R diagnostics	Skeptic
Lost variables in Dataverse export	`pctcath` variable missing from `.tab` export	Detect and flag as data-source limitation	Skeptic
R backtick quoting	Column `_computed_outcome` quoted as `‘_computed_outcome‘` by R internals	Four-way lookup: raw → backtick → strip → fuzzy (9 call sites)	R core
Destructive `gen` safeguard	`capture drop X; gen X = expr` deletes `X` when `expr` fails	Backup-restore pattern: rename → gen → restore on failure	Janitor
Format conversion (`.dta` → CSV)	Stata binary format unreadable by R/Python	Automatic pandas-based conversion	Runner

Class 4: Variable name mismatches. Variable names extracted from code do not always match column names in the exported analysis dataset. Discrepancies arise from typos, truncation, macro expansion, encoding differences, or computed expressions. The patterns below record the matching rules added to reconcile these differences.

Pattern	**Manifestation**	**Resolution**	**Agent**
Single-character typos	`lcri_euac1_r` vs. `lcri_euc1_r`	Four-tier resolution: exact → case-insensitive → Levenshtein ≤ 2 → prefix match	Skeptic
Missing suffixes	`serfperc` vs. `serfperc1`	Levenshtein distance matching	Skeptic
Prefix matches	`incumbvotes` vs. `incumbvotesmajorpercent`	Bidirectional prefix detection	Skeptic
Computed expressions	`zero1(infeels-outfeels)` is a function expression, not a column name	Expression detection + precomputation (min-max normalize, log transform)	Skeptic
Unresolved Stata macros	Metadata contains `$controls_z2` or `‘controls_z2’` literally	Scan `command` field for `$xxx` references; expand using macro dictionary	Profiler
Unsplit cluster variables	`"ccode year"` → R’s `make.names()` produces `"ccode.year"`	Space-based splitting normalization for cluster variables	Skeptic
Unresolved macro passthrough	`$Z`, `$C` passed to R as literal strings	Guard: detect unresolved `$macro` patterns before invoking R	Skeptic
Case inconsistencies	Variable names differ only in capitalization	Case-insensitive matching (second tier of resolution chain)	Skeptic

Class 5: Stata syntax variants. Stata syntax differs across versions and programming styles. The pipeline encountered variations in delimiter modes, legacy commands, macro conventions, and wrapper structures. The following entries summarize the syntax-related adjustments implemented in the Janitor and related agents.

Pattern	**Manifestation**	**Resolution**	**Agent**
`#delimit ;` mode	Semicolons replace newlines as statement terminators; multi-line commands	Delimiter state tracking; treat content between semicolons as one statement	Janitor
`#delimit` abbreviations	`#d`, `#delim`, `#delimi` are all valid	Extended regex: `#d(?:e(?:l(?:i(?:m(?:i(?:t)?)?)?)?)?)?`	Janitor
Old `merge` syntax	`merge cow year using "file.dta"` (pre-Stata 11)	Auto-detection and conversion to `merge m:1 ...`	Janitor
Weight specifications	`[aweight=w]`, `[pweight=w]` embedded in IV commands	Separate weight parsing before variable-list extraction	Runner

Pattern	**Manifestation**	**Resolution**	**Agent**
`e(sample)` filtering	`ivreg2 ... if e(sample)` references prior estimation's sample	Full-panel export with `janitor_esample` flag column; R filters post-lag-computation	Janitor, R core
`capture noisily` side effects	Wrapping failed commands corrupts `e(sample)` and `e(b)`	Comment out non-target IV commands entirely instead of wrapping	Janitor
`e(sample)` restoration	After `capture noisily` failure, `e(sample)` is invalid	Backward scan to find estimation command that set `e(sample)`; inject re-estimation	Janitor
User-defined commands	`edvreg` as custom `ivreg2` wrapper	Maintain list of known user-defined commands; detect `program define` blocks	Janitor
Wrapper commands	`parmby "ivreg2 ...", ...`, `bootstrap`, `jackknife`	Wrapper exclusion list: preserve IV commands inside wrappers from commenting	Janitor
Multi-line command end detection	In `#delimit ;` mode, commands span multiple lines until `;`	Extended `_find_end_of_stata_cmd()` to scan forward to semicolon in delimiter mode	Janitor

Class 6: Statistical model specification variability. The IV designs in the corpus differ in fixed effects, sub-sample conditions, clustering structures, weights, and model types. Supporting these variations required additional parsing and normalization rules. The table below lists the specification-related issues resolved.

Pattern	**Manifestation**	**Resolution**	**Agent**
Panel fixed effects	`xtivreg2, fe` requires `xtset panel time`	Detect `xtset` in metadata; add FE columns to IV formula	Skeptic
Stata `if` conditions	`if year >= 2000 & region == "east"`	Extract condition; pass through to R `--if_condition` parameter	Skeptic
R inline subsetting	`data = df[df$var == 0, ]`	Manual addition of `subset()` to metadata `data_prep`	Skeptic
RDD bandwidth subsetting	`rdrobust` computes bandwidth at runtime	Compute bandwidth from execution log; hardcode as numeric subset condition	Skeptic

Pattern	Manifestation	Resolution	Agent
Endogenous variable interactions	`log(providers) * as.factor(year)`	Identified as structural limitation; Journalist flags as "Non-Linear Model Approximation"	Journalist
Non-linear IV models	`ivprobit`, `ivtobit`	Journalist generates information box noting linear approximation	Journalist
Weighted regression	Stata `[pweight=w]`, R `weights = sample.size`	Cross-software weights extraction with fallback regardless of language	Skeptic
Multi-equation output	`ivprobit/ivtobit` produce two-line headers	Multi-line header detection and merging helper functions	Runner

Class 7: Runtime resource constraints. The datasets in the corpus range from small cross-sections to large panels exceeding one million observations. These differences create variation in memory use and runtime. The following entries summarize the constraints encountered and the safeguards implemented.

Pattern	Manifestation	Resolution	Agent
Large-dataset timeout/OOM	1.26M rows (Ritter 2016); 115K rows (Lelkes 2017)	`MAX_OBS = 100,000` sampling cap with fixed seed for reproducibility	R core
Cluster jackknife memory exhaustion	Large cluster count causes OOM	Fallback to observation-level jackknife (200 observations)	Skeptic
Code execution timeout	Some scripts exceed 600s	Configurable hard timeout (`--timeout`)	Runner
Pre-computed lag column NAs	Full-panel export has extra NAs in lag columns outside estimation sample	Compare NA counts: if pre-computed has more NAs than base, recompute from base	R core

Class 8: Graphics and interactive commands. Many replication packages assume an interactive environment with an available graphics device and user input. In a batch execution setting, such commands cause interruptions or failures. The table below records the patterns identified and the corresponding handling rules.

Pattern	**Manifestation**	**Resolution**	**Agent**
Common graphics commands	`graph twoway`, `histogram`, `plot()`, `ggplot()`, `plt.show()`	Regex-based commenting	Janitor
Rare Stata graphics	`cibplot`, `marginsplot`, `binscatter`, `spmap`	Expanded graphics command list	Janitor
Interactive commands	`pause`, `View()`, `browser()`, `input()`, `breakpoint()`	Comment out all interactive commands	Janitor
Output table packages	`modelsummary()`, `stargazer()`, `texreg()`	Comment out (unnecessary for data extraction; may fail in batch mode)	Janitor
Over-commenting false positives	Function named `estimate_plot_data` incorrectly flagged as graphics	Match complete function call patterns, not substrings	Janitor

Class 9: Data acquisition variability. Replication materials are hosted on multiple platforms with different API formats, URL conventions, and availability guarantees. Supporting these platforms required explicit handling of retrieval formats and error cases. The following entries document the acquisition-related issues encountered.

Pattern	**Manifestation**	**Resolution**	**Agent**
Multiple hosting platforms	Harvard Dataverse, GitHub, OSF, journal websites	Multi-platform API support	Librarian
Wrong dataset retrieval	Keyword search returns unrelated datasets with similar titles	PDF-first architecture: prioritize URLs from paper over search	Librarian
API format errors	Trailing slash in Dataverse API URL causes 404	Correct API URL formatting	Librarian
Deprecated R packages	`rgdal`, `rgeos`, `maptools` retired from CRAN in 2023	Comment out `library()` calls for known deprecated packages	Janitor
Incompatible R packages	`ri` package incompatible with R 4.x	Flagged as unfixable	—
Expired repository URLs	Repository taken offline or never publicly released	Multi-tier retrieval with manual-supply fallback	Librarian

Class 10: Code injection logic variability. To extract analysis datasets, the Janitor inserts export commands into the original scripts. The correct insertion depends on delimiter modes, multi-line commands, and surrounding control flow. The entries below summarize

the injection-related patterns resolved during development.

Pattern	**Manifestation**	**Resolution**	**Agent**
Injection in `#delimit ;` regions	Injected code uses newline terminators; surrounding code uses semicolons	Detect active delimiter mode; wrap injection with `#delimit cr`/`#delimit ;`	Janitor
Multi-line command truncation	Export block inserted between lines of a multi-line command	Scan forward to complete command end before inserting	Janitor
Line-number drift	Insertions shift all subsequent line numbers; index-based tracking breaks	Content-based detection (backward scan from marker) instead of index tracking	Janitor
Full-panel vs. `e(sample)` export	Some specifications need full panel for lag computation; others need only the estimation sample	Detect `esample_mode`; choose export strategy accordingly	Janitor
`parmest` block handling	Code between `parmest` and `restore` is interdependent	Comment out entire block as a unit	Janitor

E. Example of Diagnose Report

IV Diagnostics Report: Small Aggregates, Big Manipulation: Vote Buying Enforcement and Collective Monitoring

Miguel R. Rueda — *American Journal Of Political Science* — (2017)

Generated: 2026-02-16 15:30:49

Contents

1 Executive Summary **2**

2 Study Design **2**
2.1 The Causal Question 2
2.2 Variables 3

3 Replication Results **3**

4 Diagnostic Results **4**
4.1 Specification 1: e_vote_buying (Primary) — The incidence of vote buying reported by Colombian citizens. 4
4.1.1 Instrument Strength 4
4.1.2 Robust Inference 5
4.1.3 Sensitivity Analysis 7
4.1.4 IV vs. OLS Comparison 7
4.2 Specification 2: sum_vb — The total number of vote buying incidents reported across various contexts. 8
4.2.1 Instrument Strength 9
4.2.2 Robust Inference 9
4.2.3 Sensitivity Analysis 11
4.2.4 IV vs. OLS Comparison 11
4.3 Specification 3: e_vote_buying — The incidence of vote buying reported by Colombian citizens. 11
4.3.1 Instrument Strength 13
4.3.2 Robust Inference 13
4.3.3 Sensitivity Analysis 15
4.3.4 IV vs. OLS Comparison 15

5 Conclusions and Recommendations **15**

6 Technical Appendix **16**
6.1 Methods Used in This Report 16
6.2 Key References 16

This report evaluates the **robustness** of an instrumental variable (IV) analysis. Here, we focus on the strength of the IV and the robustness to different inferential methods. It is important to note that the credibility of an IV design relies on the instrument's unconfoundedness and exclusion restriction, which are often untestable.

1 Executive Summary

Spec	Outcome (Y)	Treatment (D)	Instrument (Z)	Eff. F	Rating
spec_1 ⋆	`e_vote_buying`	`lm_pob_mesa`	`lz_pob_mesa_f`	827.2	**HIGH**
spec_2	`sum_vb`	`lm_pob_mesa`	`lz_pob_mesa_f`	203.9	**LOW**
spec_3	`e_vote_buying`	`lm_pob_mesa`	`lz_pob_mesa_f`	8598.3	**HIGH**

Primary Specification (spec_1): HIGH - Results appear robust
PASS **No major concerns detected.**
Main Finding: The 2SLS estimate is **-1.4600** (p = 0.0016), which is **statistically significant** at the 5% level.

spec_2 (sum_vb): LOW - Significant concerns about validity

spec_3 (e_vote_buying): HIGH - Results appear robust

2 Study Design

2.1 The Causal Question

This study uses **instrumental variable (IV) analysis** to answer a causal question. Here's how to understand the key components:

Paper: *Small Aggregates, Big Manipulation: Vote Buying Enforcement and Collective Monitoring*

IV Strategy: The rules that determine maximum sizes of polling stations create exogenous variation in polling place size, allowing for the identification of the causal effect of polling place size on vote buying incidents.

2.2 Variables

Role	Variable	Description
Y1 ⋆	`e_vote_buying`	The incidence of vote buying reported by Colombian citizens.
Y2	`sum_vb`	The total number of vote buying incidents reported across various contexts.
Y3	`e_vote_buying`	The incidence of vote buying reported by Colombian citizens.
Treatment (D)	`lm_pob_mesa`	The average size of polling stations in terms of registered voters.
Instrument (Z)	`lz_pob_mesa_f`	The disaggregated electoral results at the polling station level.
Cluster	`muni_code`	Clustering unit for standard errors

Controls: 7 variables (`l4.margin_index2`, `l.nbi_i`, `l.own_resources`, `lpopulation`, `l.armed_actor`...)

3 Replication Results

Before running diagnostics, we first replicate the original analysis by executing the authors' code and extracting the IV coefficient. This confirms our diagnostics analyze the correct specification.

Spec	Outcome (Y)	Treatment (D)	2SLS Est.	Std. Err.	Match	Δ%
spec_1 ⋆	`e_vote_buying`	`lm_pob_mesa`	-1.4600	0.4625	excellent	0.00%
spec_2	`sum_vb`	`lm_pob_mesa`	-2.2420	1.2939	excellent	0.00%
spec_3	`e_vote_buying`	`lm_pob_mesa`	-0.9835	0.1423	excellent	0.00%

⋆ = primary specification

Match & Δ% columns: The **2SLS Est.** column shows the coefficient produced by running the authors' original source code (e.g., Stata). Our diagnostic engine independently re-estimates the same 2SLS specification in R using the extracted analysis dataset. **Match** rates the agreement between these two estimates, and **Δ%** reports the percentage difference. A close match ("excellent", $\Delta < 1\%$) confirms that the diagnostic results presented below are based on the correct specification.

4 Diagnostic Results

4.1 Specification 1: e_vote_buying (Primary) — The incidence of vote buying reported by Colombian citizens.

Outcome (Y): `e_vote_buying` | **Treatment (D):** `lm_pob_mesa` | **Instrument (Z):** `lz_pob_mesa_f`

> □ **Verified**: This diagnostic specification was cross-validated against Stata and matches exactly.

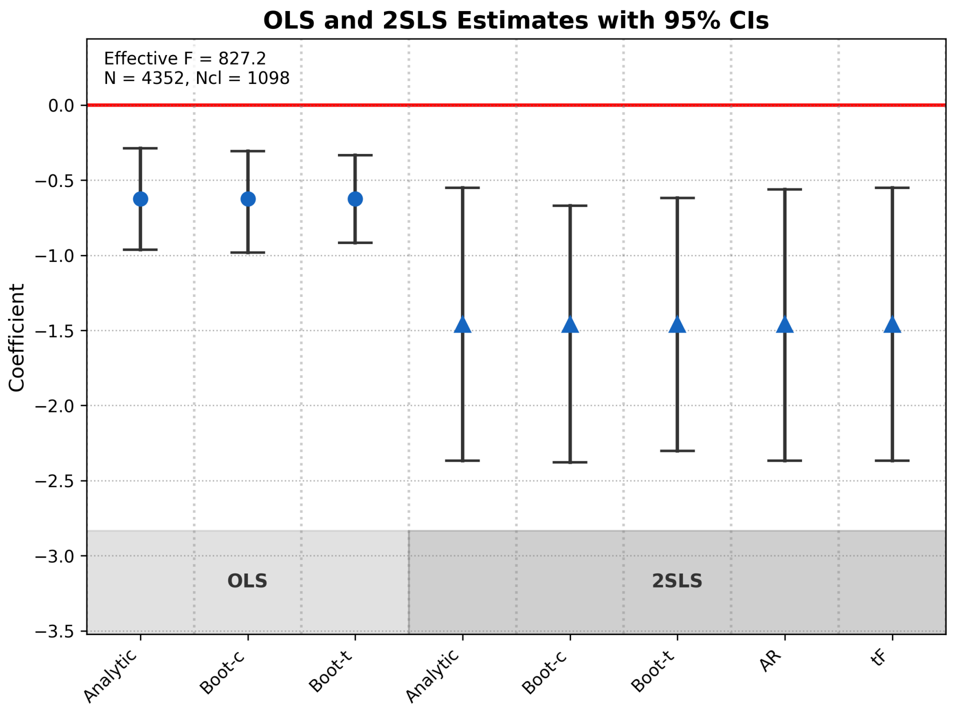


Figure 1: OLS and 2SLS estimates with 95% CIs for e_vote_buying (Primary)

4.1.1 Instrument Strength

For IV to work, the instrument must **strongly predict** the treatment. A "weak instrument" leads to unreliable estimates.

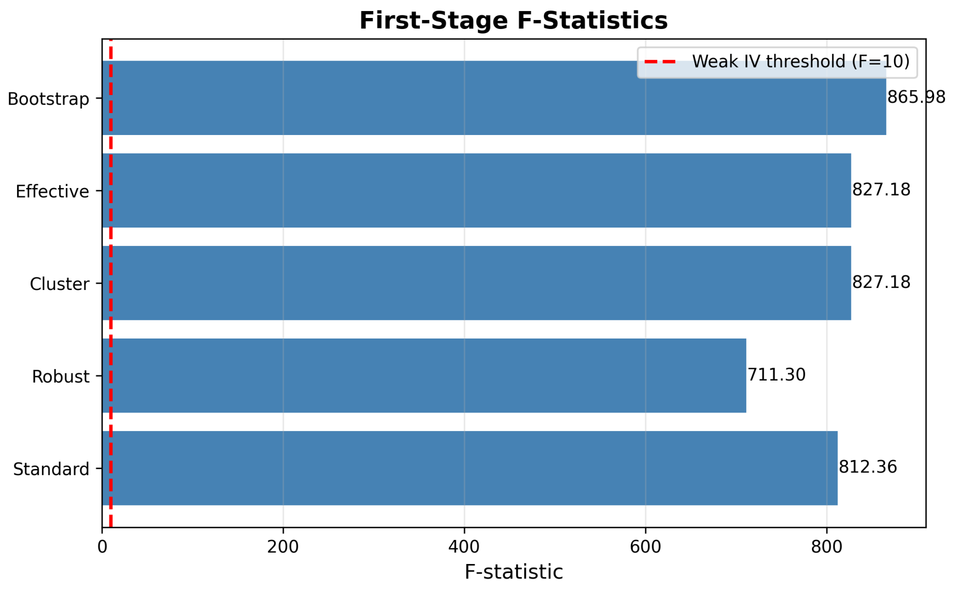


Figure 2: First-stage F-statistics for e_vote_buying (Primary)

Statistic	Value	Assessment
F-effective	827.18	**PASS** Strong
F-standard	812.36	—
F-cluster	827.18	Cluster-robust
F-robust	711.30	HC-robust
F-bootstrap	865.98	Bootstrap-robust

First-Stage Parameter	Value
First-stage coef ($\hat{\pi}$)	0.6557
First-stage SE	0.0228
First-stage ρ (correlation coefficient)	0.3970

PASS **The instrument is strong** (F = 827.18). Standard IV inference should be reliable.

4.1.2 Robust Inference

Here, we gauge the **uncertainties** of the 2SLS estimates.

Statistic	Value
Coefficient	-1.4600
Standard Error	0.4632
p-value	0.0016
95% CI	[-2.3678, -0.5521]
N	4352
N clusters	1098

A one-unit increase in treatment is associated with a 1.4600 decrease in the outcome (p = 0.0016). **Statistically significant at 5%.**

Anderson-Rubin Test (weak-IV robust): p = 0.0016 → **PASS** **Significant**
AR 95% CI: [-2.3678, -0.5614] (bounded)
tF Procedure (Lee et al. 2022): |t| = 3.15 vs critical t = 1.96 → **PASS** **Significant**
tF 95% CI: [-2.3678, -0.5521]

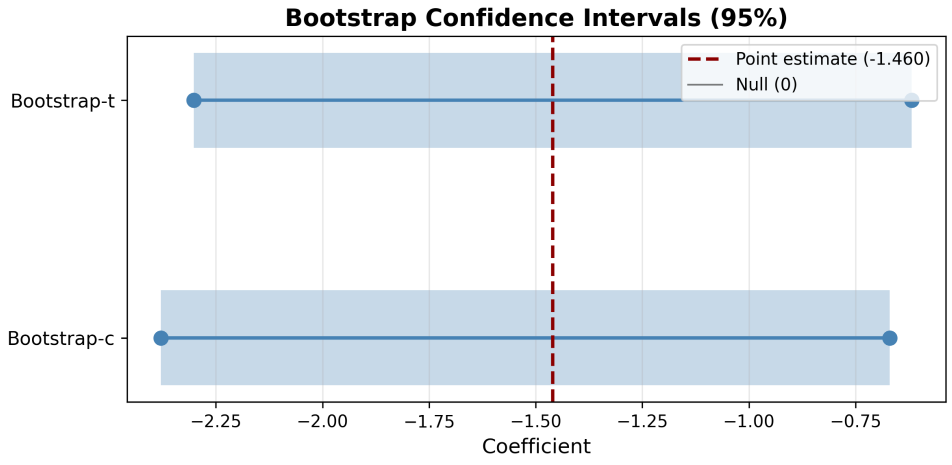


Figure 3: Bootstrap confidence intervals for e_vote_buying (Primary)

Method	95% CI	Includes Zero?
Bootstrap-c	[-2.3785, -0.6694]	No
Bootstrap-t	[-2.3017, -0.6182]	No

PASS Bootstrap CI **excludes zero** — effect is significant.

4.1.3 Sensitivity Analysis

The **jackknife** method removes each observation/cluster one at a time and re-estimates the effect. Stable results = robust findings.

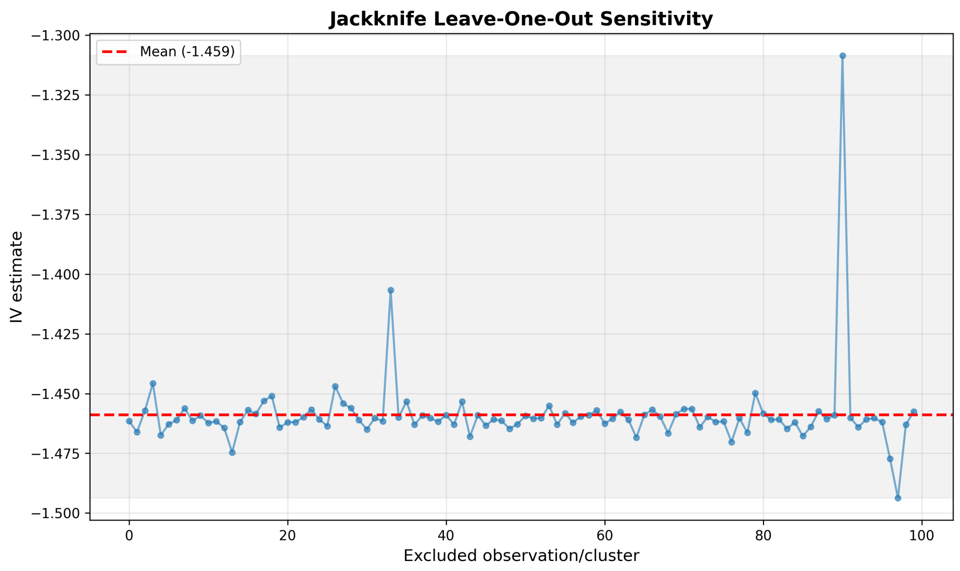


Figure 4: Jackknife leave-one-out sensitivity for e_vote_buying (Primary)

Statistic	Value
Mean estimate	-1.4589
Range	[-1.4937, -1.3086]
Std. deviation	0.0171
Most influential unit	11001 (Δ = 0.1513)

PASS **Robust** — only 12.7% variation across leave-one-out samples.

4.1.4 IV vs. OLS Comparison

Comparing 2SLS estimate to the naive OLS estimate.

Method	Coefficient	Ratio
OLS	-0.6255	—
2SLS	**-1.4600**	**2.3x**

The 2SLS estimate is 2.3x larger than the naive OLS estimate, suggesting moderate endogeneity correction.

4.2 Specification 2: sum_vb — The total number of vote buying incidents reported across various contexts.

Outcome (Y): `sum_vb` | **Treatment (D):** `lm_pob_mesa` | **Instrument (Z):** `lz_pob_mesa_f`

> □ **Verified**: This diagnostic specification was cross-validated against Stata and matches exactly.

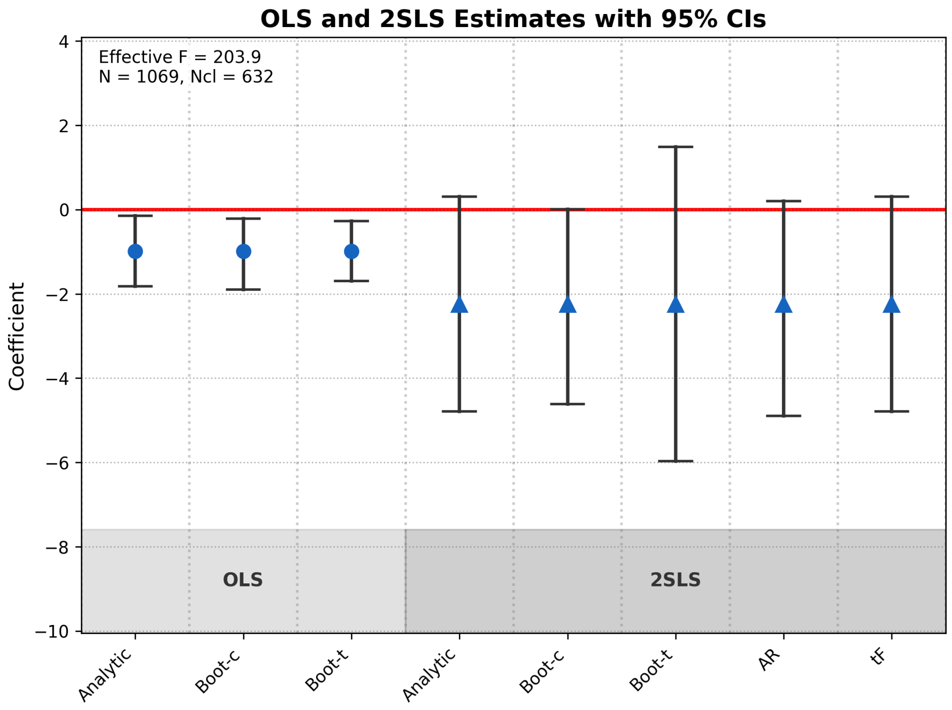


Figure 5: OLS and 2SLS estimates with 95% CIs for sum_vb

4.2.1 Instrument Strength

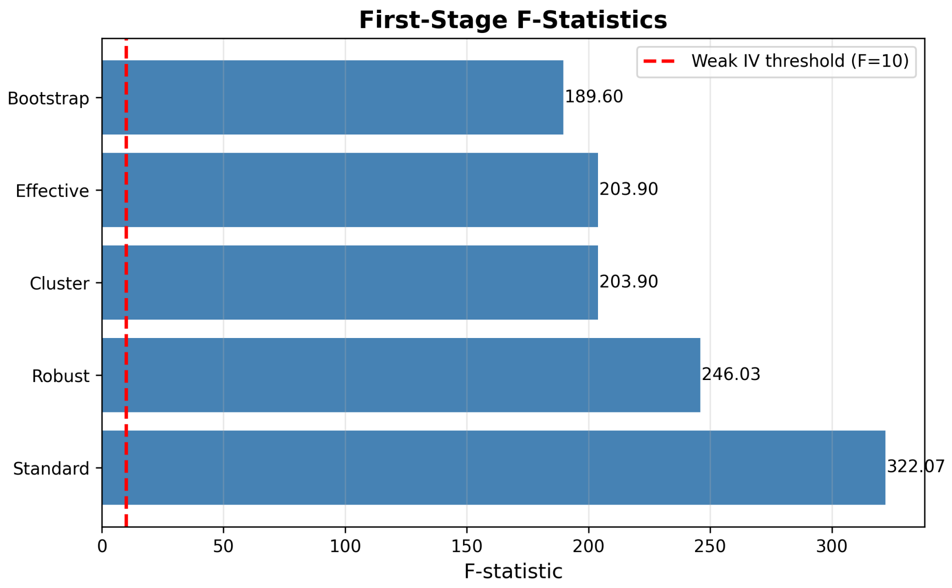


Figure 6: First-stage F-statistics for sum_vb

Statistic	Value	Assessment
F-effective	203.90	**PASS** Strong
F-standard	322.07	—
F-cluster	203.90	Cluster-robust
F-robust	246.03	HC-robust
F-bootstrap	189.60	Bootstrap-robust

First-Stage Parameter	Value
First-stage coef ($\hat{\pi}$)	0.8526
First-stage SE	0.0597
First-stage ρ (correlation coefficient)	0.4827

PASS **The instrument is strong** (F = 203.90). Standard IV inference should be reliable.

4.2.2 Robust Inference

Statistic	Value
Coefficient	-2.2420
Standard Error	1.2998
p-value	0.0845
95% CI	[-4.7895, 0.3055]
N	1069
N clusters	632

The effect is **not statistically significant (p = 0.0845 > 0.05)**.

Anderson-Rubin Test (weak-IV robust): p = 0.0747 → **FAIL** **Not significant**
AR 95% CI: [-4.8935, 0.2015] (bounded)
tF Procedure (Lee et al. 2022): |t| = 1.73 vs critical t = 1.96 → **FAIL** **Not significant**
tF 95% CI: [-4.7895, 0.3055]

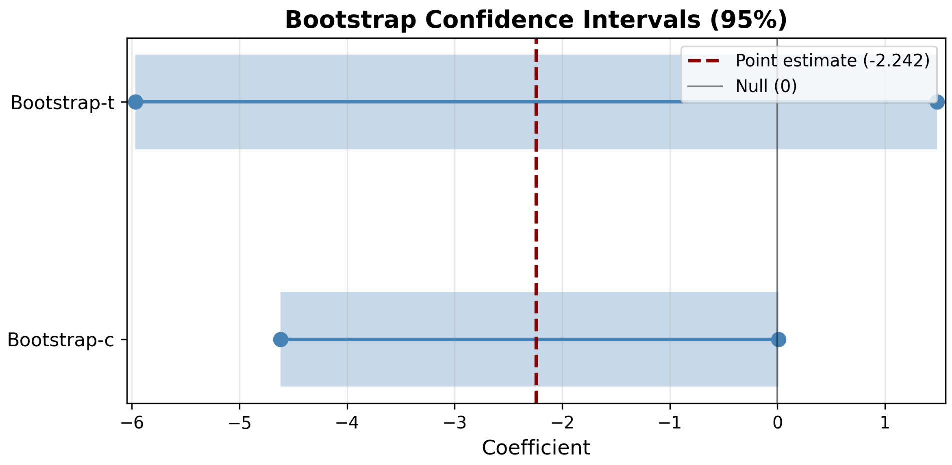


Figure 7: Bootstrap confidence intervals for sum_vb

Method	95% CI	Includes Zero?
Bootstrap-c	[-4.6198, 0.0109]	Yes
Bootstrap-t	[-5.9682, 1.4841]	Yes

WARNING Bootstrap CI **includes zero** — effect may not be significant.

4.2.3 Sensitivity Analysis

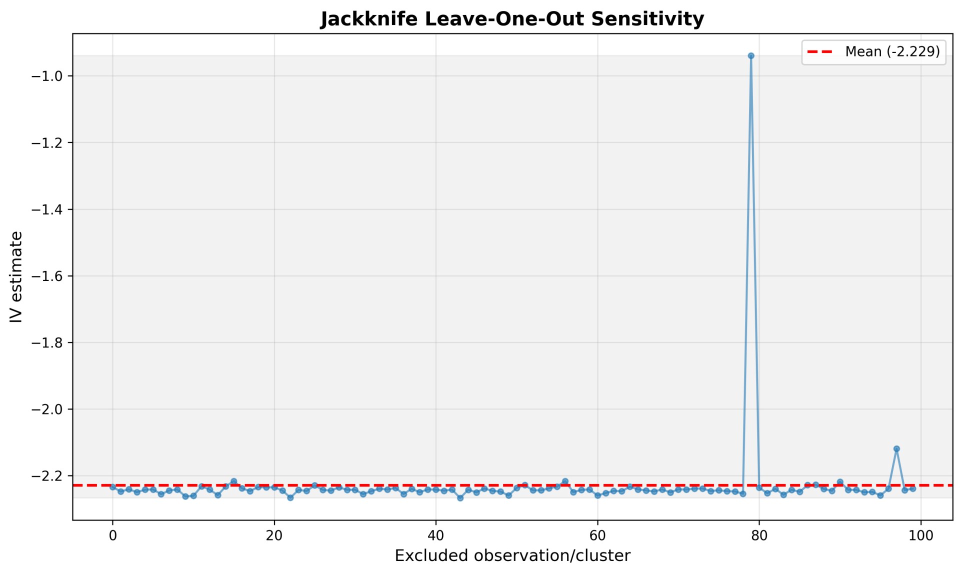


Figure 8: Jackknife leave-one-out sensitivity for sum_vb

Statistic	Value
Mean estimate	-2.2293
Range	[-2.2674, -0.9393]
Std. deviation	0.1312
Most influential unit	11001 (Δ = 1.3027)

WARNING **Sensitive** — 59.2% variation across leave-one-out samples.

4.2.4 IV vs. OLS Comparison

Method	Coefficient	Ratio
OLS	-0.9841	—
2SLS	**-2.2420**	**2.3x**

The 2SLS estimate is 2.3x larger than the naive OLS estimate, suggesting moderate endogeneity correction.

4.3 Specification 3: e_vote_buying — The incidence of vote buying reported by Colombian citizens.

Outcome (Y): `e_vote_buying` | **Treatment (D):** `lm_pob_mesa` | **Instrument (Z):** `lz_pob_mesa_f`

> □ **Verified**: This diagnostic specification was cross-validated against Stata and matches exactly.

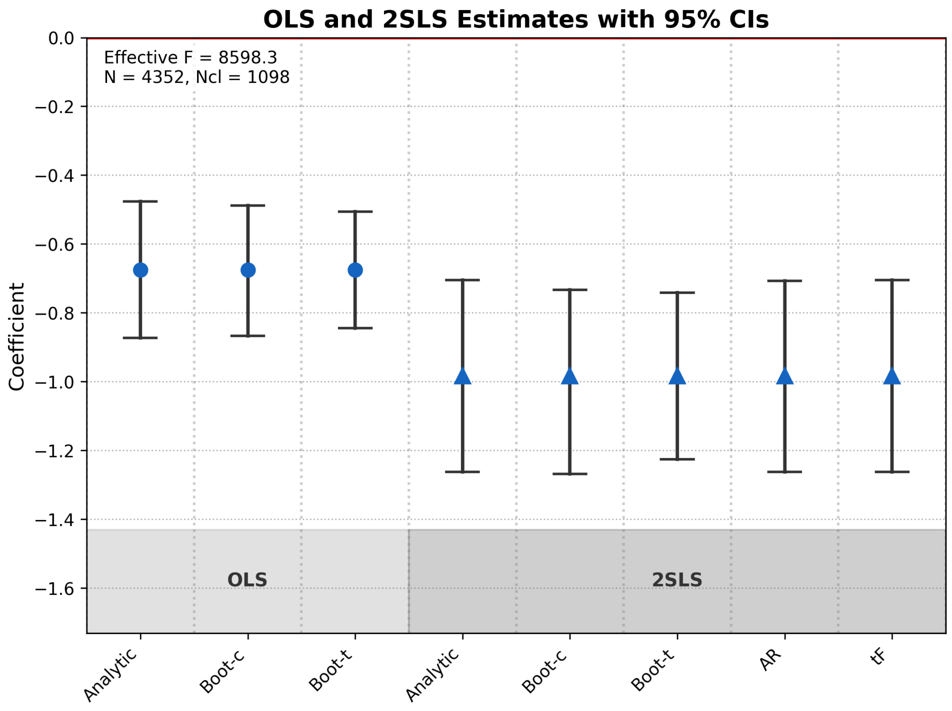


Figure 9: OLS and 2SLS estimates with 95% CIs for e_vote_buying

4.3.1 Instrument Strength

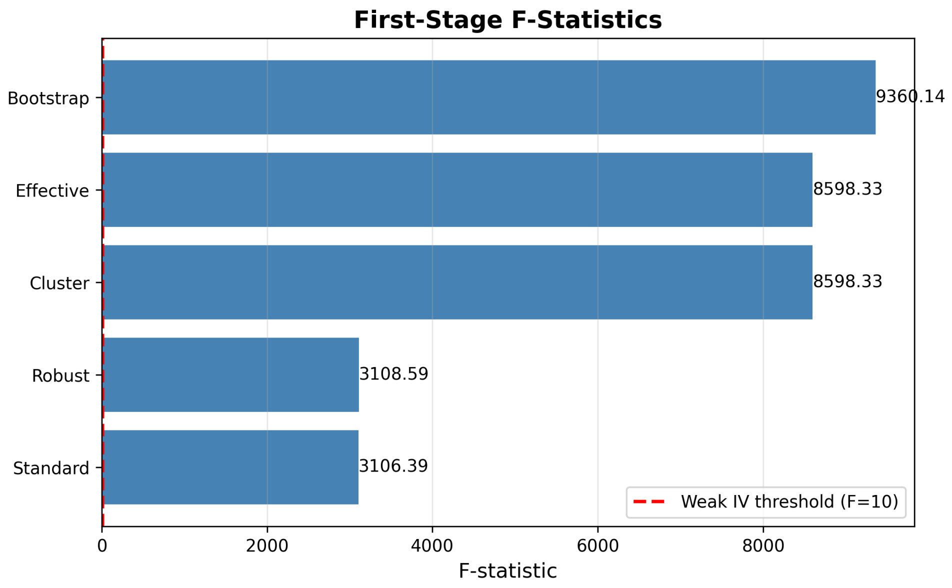


Figure 10: First-stage F-statistics for e_vote_buying

Statistic	Value	Assessment
F-effective	8598.33	PASS Strong
F-standard	3106.39	—
F-cluster	8598.33	Cluster-robust
F-robust	3108.59	HC-robust
F-bootstrap	9360.14	Bootstrap-robust

First-Stage Parameter	Value
First-stage coef ($\hat{\pi}$)	0.7957
First-stage SE	0.0086
First-stage ρ (correlation coefficient)	0.6455

PASS **The instrument is strong** (F = 8598.33). Standard IV inference should be reliable.

4.3.2 Robust Inference

Statistic	Value
Coefficient	-0.9835
Standard Error	0.1424
p-value	0.0000
95% CI	[-1.2626, -0.7044]
N	4352
N clusters	1098

A one-unit increase in treatment is associated with a 0.9835 decrease in the outcome (p = 0.0000). **Statistically significant at 5%.**

Anderson-Rubin Test (weak-IV robust): p = 0.0000 → **PASS** **Significant**
AR 95% CI: [-1.2626, -0.7073] (bounded)
tF Procedure (Lee et al. 2022): |t| = 6.91 vs critical t = 1.96 → **PASS** **Significant**
tF 95% CI: [-1.2626, -0.7044]

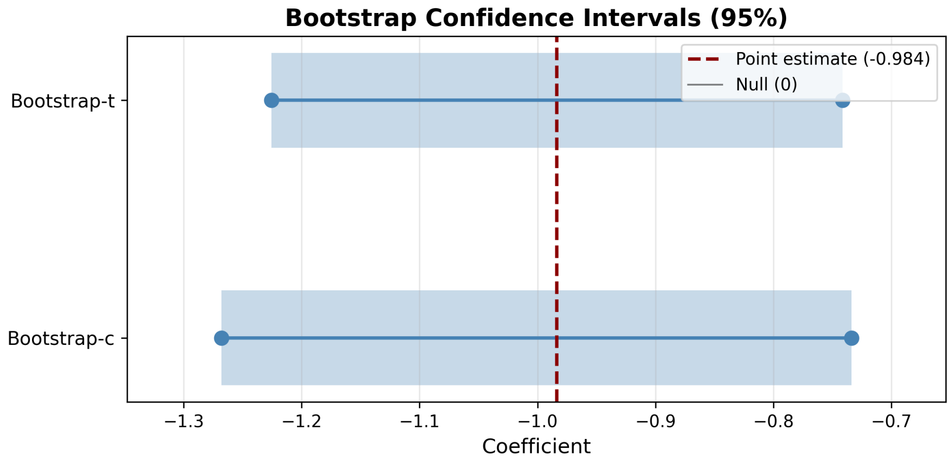


Figure 11: Bootstrap confidence intervals for e_vote_buying

Method	95% CI	Includes Zero?
Bootstrap-c	[-1.2680, -0.7339]	No
Bootstrap-t	[-1.2256, -0.7414]	No

PASS Bootstrap CI **excludes zero** — effect is significant.

4.3.3 Sensitivity Analysis

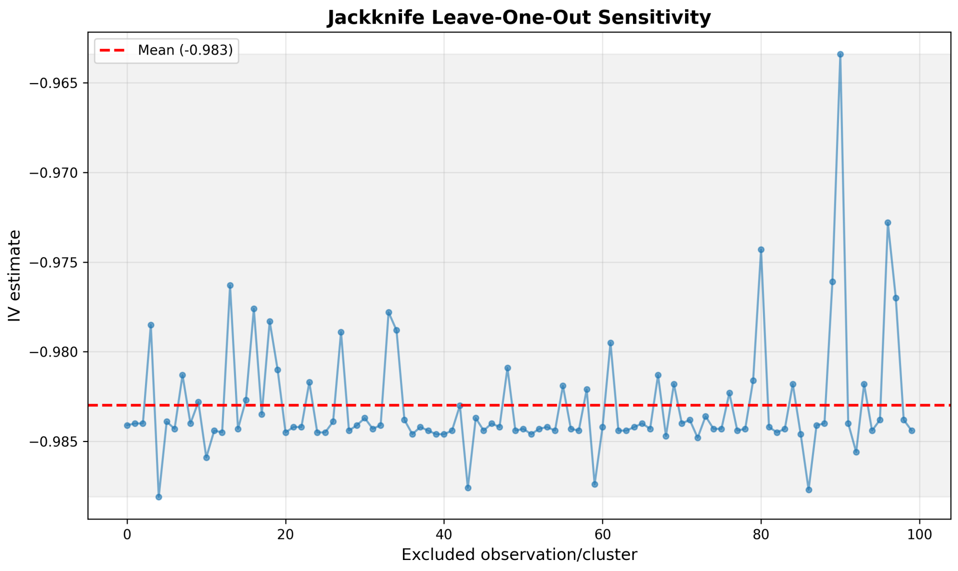


Figure 12: Jackknife leave-one-out sensitivity for e_vote_buying

Statistic	Value
Mean estimate	-0.9830
Range	[-0.9881, -0.9634]
Std. deviation	0.0033
Most influential unit	11001 (Δ = 0.0201)

PASS **Robust** — only 2.5% variation across leave-one-out samples.

4.3.4 IV vs. OLS Comparison

Method	Coefficient	Ratio
OLS	-0.6750	—
2SLS	**-0.9835**	**1.5x**

PASS The 2SLS estimate and the naive OLS estimate are **similar** (ratio = 1.46) — little evidence of bias.

5 Conclusions and Recommendations

This study was evaluated across **3 specifications**. Ratings: 2 HIGH, 1 LOW.

PASS **The IV estimates appear robust to weak IV tests and robust inferential methods.** The instrument is strong, results are significant under robust tests, and there are no major red flags.
If you believe the key identifying assumptions—namely, the instrument's unconfoundedness and the exclusion restriction—are valid, you can interpret the 2SLS estimate as causal.

6 Technical Appendix

6.1 Methods Used in This Report

Method	**Purpose**	**Reference**
F-statistic (effective)	Test instrument strength	Olea & Pflueger (2013)
Anderson-Rubin test	Weak-IV robust inference	Anderson & Rubin (1949)
Bootstrap CI (percentile)	Robust confidence intervals	Efron (1979)
Bootstrap CI (studentized)	More accurate small-sample CI	Hall (1992)
Jackknife	Sensitivity to influential obs	Quenouille (1956)
tF procedure	Weak-IV robust critical values	Lee et al. (2022)

6.2 Key References

- **Lal et al. (2024).** "How Much Should We Trust Instrumental Variable Estimates in Political Science?" *Comprehensive guide to IV diagnostics.*
- **Stock & Yogo (2005).** "Testing for Weak Instruments." *Established the $F \geq 10$ rule of thumb.*
- **Olea & Pflueger (2013).** "A Robust Test for Weak Instruments." *Effective F-statistic for heteroskedastic errors.*
- **Lee et al. (2022).** "Valid t-ratio Inference for IV." *tF procedure for weak instrument inference.*

Report generated by Journalist Agent (IV Replication Workflow)