A robustness-first framework for reproducible computational research pipelines: a simulation study

Maya Chen*[1], Daniel Okafor[2] and Sofia Alvarez[1,3]

[1]*Centre for Open Methods, Northbridge University, Northbridge, UK*

[2]*Department of Statistics, Lakeview Institute of Technology, Toronto, Canada*

[3]*Research Software Group, Instituto del Pacifico, Valparaiso, Chile*

Article type: Original Article

Abstract

Computational results may fail to reproduce when software dependencies, execution environments, or random processes are incompletely specified. We developed a robustness-first framework that evaluates a research pipeline under controlled environmental perturbations rather than testing it only in its original environment. A simulation study compared three documentation strategies—fully pinned, partially specified, and unpinned—across 300 synthetic analysis projects and 100 perturbed rebuilds per project. The primary outcomes were successful execution and numerical deviation from a reference result. Fully pinned projects reproduced successfully in 96.7% of rebuilds, compared with 82.4% for partially specified projects and 61.3% for unpinned projects. Environment capture reduced numerical deviation, while explicit random seeds chiefly reduced variation among successful runs. These effects were complementary rather than interchangeable. The framework turns reproducibility from a one-time claim into a measurable stress test and provides a concise reporting structure for authors, reviewers, and research-software teams. The study is illustrative, but the workflow can be adapted to empirical pipelines in any computational discipline.

*Corresponding author: maya.chen@example.edu

Keywords: computational reproducibility; dependency management; open science; robustness; simulation study

Subject classification codes: 68N30; 62J05

1 Introduction

Reproducibility is a foundation of cumulative computational research, yet the term describes several distinct goals. A result may be repeatable in the original environment, reproducible from shared code and data, or robust to reasonable changes in software and hardware. Treating these goals as a single binary property can conceal important failure modes (Peng, 2011; Stodden et al., 2016). For example, a script may execute today but fail after an indirect library is updated, or it may run successfully while producing a materially different estimate because a random seed was not recorded.

Practical guidance emphasizes version control, scripted analyses, preserved raw data, and recorded intermediate results (Sandve et al., 2013). The FAIR principles similarly stress that digital research objects should be findable, accessible, interoperable, and reusable (Wilkinson et al., 2016). These practices improve the conditions for reuse, but they do not by themselves quantify how resilient a pipeline is to environmental change. A robustness test is therefore a useful companion to a reproducibility checklist.

This study presents a small, transparent framework for stress-testing computational pipelines. We ask three questions:

1. How strongly does dependency specification affect execution success?
2. Do environment capture and random-seed control address the same failure mode or complementary ones?
3. Which compact set of measures communicates pipeline robustness without obscuring the underlying failure mechanisms?

The resulting manuscript also demonstrates the expected structure of a general Taylor & Francis submission, including author affiliations, an abstract, keywords, equations, figures, tables,

data and code statements, contributor roles, disclosure, funding, and an author–date reference list.

2 Methods

2.1 Study design

We generated 300 synthetic analysis projects, each representing a common read–transform–model–report workflow. Every project contained a tabular input, a deterministic preprocessing stage, a regression model, and a rendered summary report. Projects were allocated equally to three environment-documentation strategies:

Fully pinned

Direct and transitive dependency versions, operating-system metadata, and a cryptographic lock file were recorded.

Partially specified

Direct dependencies and their minimum versions were recorded, but transitive versions and system libraries were allowed to vary.

Unpinned

Package names were recorded without version constraints; the most recent compatible packages were selected at rebuild time.

Each project was rebuilt 100 times under perturbations representing dependency updates, changes in system libraries, and processor-level numerical variation. Half of the projects in each strategy recorded an explicit random seed. The simulation therefore contained 30,000 rebuild attempts. Figure 1 summarizes the design.

2.2 Outcomes

The primary binary outcome, execution success, indicated that the pipeline completed and produced every declared output. For successful rebuilds, numerical deviation measured the

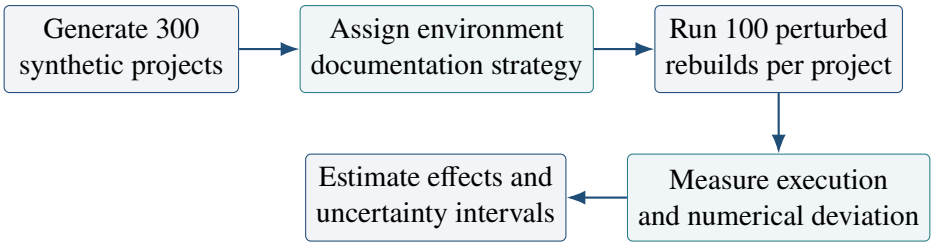

Figure 1. Robustness-testing workflow used in the simulation study. Every synthetic project was evaluated repeatedly under controlled environmental perturbations.

scaled absolute difference between a rebuilt estimate $\hat{\theta}_{ir}$ and the project's reference estimate $\hat{\theta}_{i0}$:

$$d_{ir} = \frac{\left|\hat{\theta}_{ir} - \hat{\theta}_{i0}\right|}{\max\left(\left|\hat{\theta}_{i0}\right|, \epsilon\right)}, \qquad \epsilon = 10^{-8}, \tag{1}$$

where i indexes projects and r indexes rebuilds. We report $100 d_{ir}$ as a percentage. A rebuild was classified as substantively reproduced when it both executed successfully and had $d_{ir} < 0.01$.

2.3 Statistical analysis

Execution success was summarized by strategy with project-clustered bootstrap intervals based on 2,000 resamples. A logistic regression estimated contrasts between documentation strategies. Among successful rebuilds, a log-linear model estimated associations between numerical deviation, environment strategy, and seed control. We included their interaction to test whether the two safeguards were redundant. All intervals are two-sided 95% uncertainty intervals. Because the study used synthetic projects and no human or animal data, institutional ethics review was not applicable.

3 Results

3.1 Execution success

Fully pinned projects completed successfully in 96.7% of perturbed rebuilds. Success decreased to 82.4% under partial specification and 61.3% without version constraints (Table 1). The largest class of failures in the unpinned condition was an incompatible transitive dependency; system-library mismatches were the next most common. Figure 2 shows that the ranking remained

Table 1. Rebuild outcomes by environment-documentation strategy. Intervals are project-clustered 95% bootstrap intervals. Numerical deviation is summarized only among successful rebuilds.

Strategy	Success (%)	95% interval	Median deviation (%)	Reproduced (%)
Fully pinned	96.7	95.8–97.5	0.8	92.1
Partially specified	82.4	80.6–84.0	4.7	70.6
Unpinned	61.3	58.9–63.7	12.9	43.8

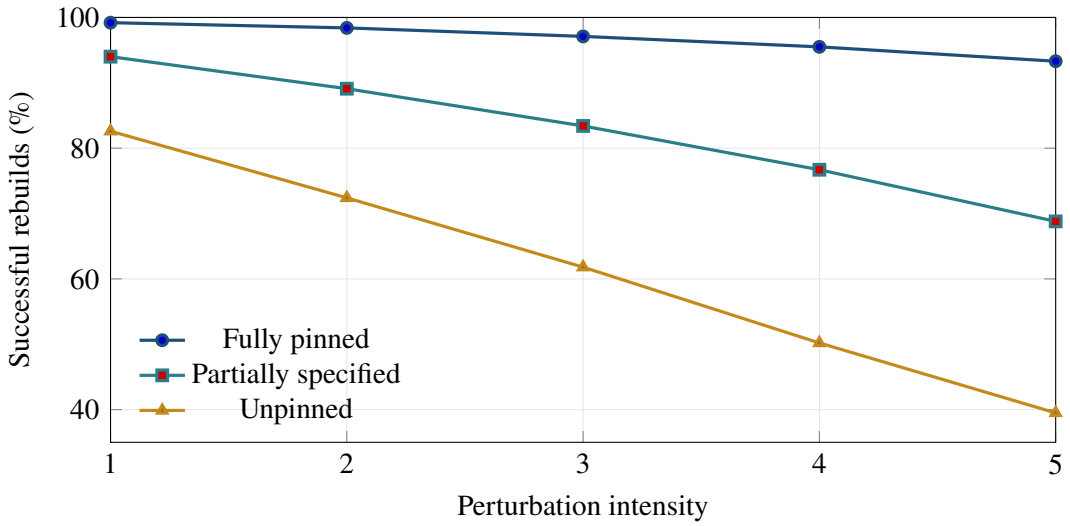

Figure 2. Execution success across increasing environmental perturbation. Points are simulation means; connecting lines aid visual comparison.

stable as perturbation intensity increased.

3.2 Complementary safeguards

Environment capture and seed control affected different outcomes. Pinning dependencies increased the probability that a pipeline completed, whereas seed control primarily reduced dispersion among pipelines that did complete. In the fully pinned condition, recording a seed reduced median numerical deviation from 1.3% to 0.4%. In the unpinned condition, the corresponding reduction was from 14.1% to 11.8%, because dependency changes dominated the remaining variation. The interaction estimate was small and its interval included zero, supporting a complementary rather than substitutive interpretation.

4 Discussion

The simulation illustrates why a successful rerun in the original environment is an incomplete reproducibility test. Pipelines with fully captured environments were substantially more resilient to dependency and system perturbations, and explicit seed control improved numerical agreement within each environment strategy. A compact evaluation should therefore report at least three elements: execution success, numerical agreement, and the conditions under which both were measured.

These findings align with calls to treat openness and reproducibility as observable practices rather than aspirations (Nosek et al., 2015). The proposed framework adds a stress-testing layer: authors declare expected outputs, rebuild the analysis under controlled changes, and report both failures and deviations. This approach can be integrated into continuous-integration systems and repeated when dependencies, data, or analysis code change.

4.1 Limitations

The study uses synthetic projects and stylized perturbations, so its numerical estimates should not be interpreted as prevalence estimates for published research. The simulated pipelines were also smaller than many production workflows and did not represent specialized hardware, restricted data, or interactive software. Empirical validation should sample real projects across disciplines, define field-specific tolerances, and distinguish failures caused by missing infrastructure from those caused by insufficient documentation.

4.2 Implications for authors and reviewers

For authors, the main practical implication is to preserve both the environment and the sources of stochastic variation. For reviewers, a failed rebuild is most informative when the submission records the attempted environment and the first divergent stage. For journals, a short machine-readable manifest could accompany the conventional data and code availability statements without imposing a single software platform.

5 Conclusion

A robustness-first assessment reveals failure modes that a one-time successful rerun cannot detect. In this simulation, version pinning protected execution, while random-seed control improved agreement among completed rebuilds. Reporting both outcomes offers a clearer and more actionable account of computational reproducibility. The framework is deliberately lightweight and can be extended to domain-specific data, software, and tolerance requirements.

Acknowledgements

The authors thank the Northbridge Open Methods reading group for comments on an earlier version of the reporting framework.

Disclosure statement

The authors report no potential conflict of interest.

Funding

This illustrative study received no external funding.

Data availability statement

The manuscript describes a synthetic demonstration. All values required to recreate its tables and figures are included directly in this source file. A real submission should replace this statement with a persistent repository link and any access conditions that apply to the underlying data.

Code availability statement

The LaTeX source is self-contained and compiles with a standard TeX Live distribution. No external analysis code is required for the illustrative results. A research article should cite an

archived release of its analysis code and record the software environment used to produce the reported outputs.

Author contributions

Maya Chen: conceptualization, methodology, writing—original draft, and visualization. Daniel Okafor: formal analysis, validation, and writing—review and editing. Sofia Alvarez: software, supervision, and writing—review and editing. All authors approved the final manuscript.

Ethics statement

This simulation study used no human participants, animals, personal data, or clinical records; ethics approval and informed consent were therefore not applicable.

Notes on contributors

Maya Chen studies transparent computational methods and research software sustainability. **Daniel Okafor** develops statistical methods for evaluating complex data-analysis workflows. **Sofia Alvarez** leads research-software engineering projects focused on durable and reusable scholarly outputs.

References

Nosek, B. A., Alter, G., Banks, G. C., et al. (2015). Promoting an open research culture. *Science*, 348(6242), 1422–1425. doi:10.1126/science.aab2374.

Peng, R. D. (2011). Reproducible research in computational science. *Science*, 334(6060), 1226–1227. doi:10.1126/science.1213847.

Sandve, G. K., Nekrutenko, A., Taylor, J., and Hovig, E. (2013). Ten simple rules for reproducible computational research. *PLoS Computational Biology*, 9(10), e1003285. doi:10.1371/journal.pcbi.1003285.

Stodden, V., McNutt, M., Bailey, D. H., Deelman, E., Gil, Y., Hanson, B., Heroux, M. A., Ioannidis, J. P. A., and Taufer, M. (2016). Enhancing reproducibility for computational methods. *Science*, 354(6317), 1240–1241. doi:10.1126/science.aah6168.

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., et al. (2016). The FAIR guiding principles for scientific data management and stewardship. *Scientific Data*, 3, 160018. doi:10.1038/sdata.2016.18.

A Minimal robustness-reporting checklist

Table 2 provides a compact checklist that can accompany the main manuscript or supplementary material.

Table 2. Suggested information for a computational robustness report.

Item	Information to report
Reference environment	Operating system, language runtime, direct and transitive dependencies, and hardware requirements.
Declared outputs	Files, tables, figures, or numerical estimates that constitute a successful run.
Perturbations	Dependency updates, system changes, randomization, and the rationale for their tested ranges.
Agreement criterion	Numerical or qualitative tolerance used to judge a rebuilt result against its reference.
Failure record	Stage of failure, diagnostic message, attempted remedy, and whether the failure is deterministic.
Preservation	Persistent identifiers for data, code, environment files, and the exact release evaluated in the manuscript.