Experience with Reproducibility and Consistency in Writing an Academic Paper

Joseph Wonsil
University of British Columbia
Vancouver, British Columbia, Canada
jwonsil@student.ubc.ca

Nichole Boufford
Oracle Labs
Vancouver, British Columbia, Canada
nichole.boufford@oracle.com

Margo Seltzer
University of British Columbia
Vancouver, British Columbia, Canada
mseltzer@cs.ubc.ca

Abstract

The iterative processes of writing code for an analysis and writing a paper based on that analysis often occur synchronously. Although this process is not inherently a problem, it can lead to inconsistencies between the data, the figures in the paper, and the prose that discusses those charts. We present our experience writing a paper that achieves consistent and reproducible results using standard tools from the reproducibility literature. We report the lessons learned from this experience and note that, while it adds additional upfront work and some mental overhead, we succeeded in satisfying our requirements. We then propose a research agenda to generate improved tools that will make these techniques widely accessible.

Keywords

reproducibility, provenance

ACM Reference Format:
Joseph Wonsil, Nichole Boufford, and Margo Seltzer. 2025. Experience with Reproducibility and Consistency in Writing an Academic Paper. In *ACM Conference on Reproducibility and Replicability (ACM REP '25), July 29–31, 2025, Vancouver, BC, Canada.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3736731.3746136

1 Introduction

The reproducibility literature focuses on making data processing and analysis reproducible. However, the challenges in ensuring the corresponding paper accurately describes the results are often overlooked. Data analysis using code and writing the paper that describes such a data analysis are both iterative processes. Errors can appear, because these two iterative processes occur synchronously. Suppose an author included two figures in a paper using results from the same computational analysis. They later modify the analysis and update "Figure 1" accordingly. By modifying the analysis, they might have unknowingly changed the result in "Figure 2" without updating the figure in the paper. In this scenario, a *result* is any text, figure, table, or other output generated from an analysis or manually compiled from external sources such as other papers. The results in their paper are no longer consistent; different executions of different versions of their analysis generated each result.

ACM REP '25, Vancouver, BC, Canada

ACM ISBN 979-8-4007-1958-5/25/07
https://doi.org/10.1145/3736731.3746136

To avoid these pitfalls while writing a now-published reproducibility paper, we set three requirements for our "analysis to paper pipeline" to ensure we lived up to the reproducibility expectations we described in our paper.

- **R.1)** Whenever we build our paper, any referenced result should be up-to-date and consistent with the most recent run of the analysis. This requirement implies that for any script that produces multiple results, if we make any changes to the script for one result, when we re-execute it, *all* the results it produces are updated in the paper.
- **R.2)** Anywhere we reference a result while writing the paper should be distinguishable from the rest of the prose.
- **R.3)** The pipeline should be able to execute on others' machines with minimal effort.

We chose to build our own pipeline, because we believed that there was not a single tool that would satisfy all our requirements. Several tools came close, but ultimately did not satisfy our requirements nor catered well to the venue's guidelines.

We constructed our pipeline out of tools that computer scientists, Linux users, and those familiar with reproducibility are likely to find familiar. We used GNU Make (Make) [4] to statically track dependencies, execute the data analysis, and build our paper. This approach satisfies R1 by ensuring a consistent and repeatable order of executions to go from data to paper. It also partially satisfies R2 as it requires saving our results to intermediate files tracked by Make that we identify using macros in our paper. We finish satisfying R2 by writing Python [1] helpers to assemble the pipeline. We satisfy R3 by creating a Docker container [3] that we use for both the development and push-button execution of our pipeline. All these components living in a single git repository backed up on GitHub ensured that it was trivial to share our setup with anyone who had access to a machine, git, Docker, and the internet.

We identify cases where these tools worked well, where there is potential for significant improvement, and problems that remain.

2 Implementation

Our pipeline initially consisted of a Python data analysis hosted in a Jupyter notebook and a paper draft written in Markdown, both saved into one Git repository. The submission venue required submissions written in Microsoft Word, but as authors attempting to follow good data management practices, we wrote the paper in Markdown and then "built" a Word document using pandoc[1]. Writing our paper in Markdown meant that our paper could be version-controlled in the same git repository as our analysis, which helped keep the two components consistent.

[1] pandoc.org

Make helped satisfy R1 by ensuring that any time we modified our analysis, the script would re-execute and save the results. We set the Make target for our final paper to depend on the contents of the directories that contained our results; if a single number in any of the results changed, the whole paper was recompiled. Whenever we changed our notebook and then tried to build the paper, Make automatically converted the notebook to a single script, re-created the analysis script, re-executed the script, and then updated all the results in the paper before recompiling it, thus satisfying R1.

We wrote any computation that Make could not perform in Python helpers, such as macro-substitution to connect the analysis and the paper. One helper ensured data were saved to the directories declared as dependencies in our Makefile. In the paper, we refer to these macros instead of the result itself. Searching for the macro identifiers allowed us to quickly identify everywhere that we reference a result, satisfying R2. A Python script then read the files in the output directories and replaced any instance of the "macro" with the matching file's contents. We used this process to insert all textual results into our paper.

Since using Python introduced a dependency problem for reproducibility, we placed a Dockerfile that creates a container capable of executing everything from start to finish in our pipeline into our main directory. Using Docker ensured that we could execute across multiple devices, satisfying R3.

3 Lessons learned

Our extended pipeline simplified the mental overhead required to keep our paper consistent and reproducible at the cost of upfront work. Once we built the pipeline, it required minimal maintenance. Future users, such as ourselves, can avoid a significant amount of upfront overhead by using our project as a template.

At the start, we viewed our upfront work as "insurance" and wondered if it would pay off - and it did. At one point, we changed how we framed the discussion of the results we were reporting, i.e., rather than minimize errors, we maximized successes. Our setup allowed us to quickly identify where we had numbers in the prose that needed updating to reflect this change. Then, months after its completion, we were able to pick up where we left off when we were writing a response to the editors. Most of all, we felt comfortable with the reproducibility and consistency of our results, since our numbers always came from the same execution in a portable environment. However, the automatic execution of our analysis did reveal a startling new fear: numbers we refer to could change without our knowledge, causing a disconnect between their reality and the prose. We realized that while we are heading in the right direction, more work remains to be done if we want the *paper itself* to be cohesive and always accurate. Therefore, if the goal is to ensure that a paper is always consistent, this toolchain helps, but it cannot guarantee that your prose matches your results.

Our experience revealed that our pipeline could use more detailed provenance tracking. Using the macros, we can identify where we use results, but sometimes we stop using some of these results, causing bloat in our directories. However, we also noticed that if we stop generating a result from our analysis, the saved result will still exist even though the analysis no longer writes to it every time it executes. Because it would still be substituted into the paper by our Python script, we would have a false guarantee that it came from the most recent run. We could identify these unnecessary data in the directory with more detailed provenance tracking.

A more serious issue for our system is that it does not report errors or warnings. A particularly interesting error is one that creates a mismatch between prose and numerical results. Imagine that a change to a script changes results in ways that we did not anticipate, e.g., producing results for our system that were originally better than a baseline comparison but no longer are. Suddenly the paper's claims that our system outperformed the baseline would be wrong! With our tool, it is theoretically easier for an editor to notice that, for example, a number comparison no longer makes sense ("Why do we claim 94% recall is better than 96%!?"). The simplest solution is to generate a diff of the paper before each compilation. This diff could bring changes to the attention of any diligent editor. Unfortunately, this solution will not help with results that are *indirectly* referenced. For example, the paper might only say, "and we outperform the state of the art" in the introduction.

4 Related Work

The closest existing tool to our makeshift pipeline is Madagascar [2]. It is an analysis-to-paper pipeline, originating from geophysics. Madagascar uses SCons, which has properties they found more desirable than Make such as tracking changes using MD5 hashes. Madagascar could replace Make in our pipeline, as they built extensions to SCons that facilitate connecting analyses to papers. However, it still has most of the same flaws as our pipeline.

Executable papers are interactive programs that include datasets and code to validate results alongside a publication [5]. This process requires less provenance tracking to connect the code to the prose, as there is a direct connection by construction. Unfortunately, it is challenging to submit executable papers when many venues have set submission formats (e.g., Latex or Microsoft Word).

5 Conclusion

Our experience developing our pipeline reveals that writing consistent and reproducible papers becomes more attainable with careful proactive effort. We constructed a reliable pipeline from data analysis to final publication with existing tools that satisfied our requirements and, once assembled, was easy to maintain. However, it is imperfect, as there is no detailed provenance tracking for the files used between the analysis and the paper. We also prefer, in the future, to have a way to run automatic checks or tests on the results to make sure that they support the claims in the prose and have not changed too drastically.

References

[1] [n. d.]. The Python Language Reference. https://docs.python.org/3/reference/index.html.
[2] Sergey Fomel, Paul Sava, Ioan Vlad, Yang Liu, and Vladimir Bashkardin. 2013. Madagascar: Open-Source Software Project for Multidimensional Data Analysis and Reproducible Computational Experiments. *Journal of Open Research Software* 1, 1 (Nov. 2013), e8–e8. doi:10.5334/jors.ag
[3] Dirk Merkel. 2014. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux Journal* 2014, 239 (March 2014), 2:2.
[4] Paul D. Smith Richard M. Stallman, Roland McGrath. 2020. GNU Make. Free Software Foundation.
[5] Rudolf J Strijkers, Reginald Cushing, Dmitry Vasyunin, Cees de Laat, Adam Belloum, Robert J Meijer, et al. 2011. Toward Executable Scientific Publications.. In *ICCS*. 707–715.