

Reproducible Scientific Workflow Generation through Agentic Code Verification

Authors

Eleanor Hartley^{1*}, James R. Whitfield², Olivia M. Bennett³

Affiliations

Department of Computer Science, The University of Manchester, Kilburn Building, Oxford Road, Manchester M13 9PL, United Kingdom

*Corresponding author: e.hartley@manchester.ac.uk

Abstract

Scientific data analysis increasingly depends on code, but manually written analysis scripts often suffer from missing preprocessing steps, inconsistent statistical tests, undocumented parameters, and poor reproducibility. Large language models can generate analysis code from natural-language research questions, yet the generated workflows may execute incorrectly or produce results that cannot be reproduced across environments. This study explores reproducible scientific workflow generation through agentic code verification. We propose SciFlow-Agent, a multi-agent model containing a data inspection agent, a statistical planning agent, a workflow code generation agent, a visualization agent, and a reproducibility verification agent. The data inspection agent identifies variable types, missing values, outliers, and table relationships. The statistical planning agent selects suitable methods, including regression, survival analysis, time-series modeling, hypothesis testing, and non-parametric comparison. The workflow code generation agent produces executable Python and R scripts. The visualization agent generates publication-ready figures, while the verification agent reruns the workflow under controlled environments and checks numerical consistency, dependency completeness, and result traceability. Experiments were conducted on 740 public datasets from biomedical research, environmental science, public health, economics, and social science. The benchmark included 2,960 analysis tasks, 8,420 expected statistical outputs, and 5,180 figure-generation requirements. Compared with a single-agent code generation model, SciFlow-Agent improved full-workflow execution success from 57.3% to 79.5%. Reproducible result matching increased from 61.8% to 84.2%, and statistical-method selection errors decreased by 36.7%. The average absolute deviation between regenerated numerical results and reference outputs was reduced from 0.084 to 0.031. Dependency-related failures decreased by 48.6%, and figure specification accuracy improved from 70.4% to 86.9%. These results suggest that agentic code verification can strengthen the reliability, transparency, and reproducibility of automated scientific programming.

Keywords: Scientific workflow; code generation; reproducibility; statistical computing; automated data analysis; code verification; multi-agent systems

1. Introduction

Scientific data analysis increasingly depends on executable computational workflows for data inspection, preprocessing, statistical testing, model fitting, sensitivity analysis, visualization, and result reporting. These operations are often implemented through Python, R, or mixed-language pipelines, and each analytical decision should be sufficiently documented so that another researcher can reproduce the reported results. Jupyter notebooks provide a convenient environment for combining executable code, explanatory text, intermediate outputs, and figures, but notebook-based organization alone does not guarantee reproducibility across computers or software environments [1]. Empirical studies of shared R scripts, biomedical notebooks, and computational research repositories have identified recurring problems such as undeclared packages, unavailable input files, hidden execution states, inconsistent cell order, and insufficient documentation of analytical decisions [2]. Reproducibility guidelines therefore emphasize complete analysis scripts, version-controlled code, declared software environments, accessible source data, and traceable records of data transformation and model configuration [3]. More broadly, recent work on globally informed computational pruning has shown that redundant processing can be reduced when a system identifies important computational units using information from the full task context rather than making isolated local decisions [4]. A related principle is relevant to scientific workflows, where verification effort should be concentrated on steps that can materially affect statistical validity, numerical consistency, or the interpretation of results. These requirements are not limited to a single discipline but apply broadly to biomedical, environmental, public-health, economic, behavioral, and social-science research [5,6].

Large language models have introduced a new mechanism for constructing such analytical workflows from natural-language research questions. Early code-oriented models demonstrated that large pretrained language models could generate executable programs, complete partially written functions, and support code understanding [7,8]. Codex showed strong performance on natural-language-to-code generation [9], while CodeT5 incorporated code-specific pretraining objectives for generation and understanding tasks [10]. CodeGen and AlphaCode extended this direction toward longer synthesis processes and more difficult programming problems [11,12]. More recent models, including Code Llama and DeepSeek-Coder, have further improved accessibility and performance for code generation, completion, explanation, and repair [13,14]. These advances create clear opportunities for scientific computing because researchers can increasingly describe an intended analysis in natural language and obtain executable data-processing or statistical code without manually implementing every operation. Scientific analysis, however, imposes requirements that differ substantially from conventional programming benchmarks. A script may execute without raising an exception yet remain scientifically inappropriate. It may apply a parametric test without checking relevant assumptions, ignore missing-data mechanisms, encode categorical variables incorrectly, introduce information leakage during model fitting, or apply multiple comparisons without suitable correction. A workflow may

also generate different numerical results when package versions, random seeds, data ordering, or preprocessing defaults change. Similarly, a technically correct plotting command may produce a figure that does not satisfy the analytical objective, omit uncertainty information, or represent transformed variables inconsistently with the statistical model [15,16]. Functional execution is therefore only one component of scientific correctness. Reliable computational analysis also requires consistency between the research question, data characteristics, statistical assumptions, software dependencies, numerical outputs, and reported results [17,18]. Recent agent-based systems have begun to address more complex software and analytical tasks by combining planning, code generation, execution, inspection, and iterative correction. Studies have evaluated autonomous or semi-autonomous agents on repository-level software repair, test execution, data-science problems, and scientific programming benchmarks [19,20]. Related work in biostatistics and computational research has examined language-model-assisted analytical planning, workflow provenance, and the generation of executable research procedures [21,22]. Reviews of large language models in scientific practice also emphasize the importance of transparent evaluation, provenance tracking, and human verification because plausible model outputs do not necessarily correspond to valid scientific reasoning [23,24]. These studies indicate that agentic workflows can reduce manual coding effort, but they also reveal a gap between general-purpose coding success and trustworthy scientific analysis. Existing evaluations frequently focus on whether generated code passes predefined tests, returns an expected answer, or completes a data-analysis task. Such metrics provide useful evidence of functional performance but do not fully characterize scientific reproducibility. Statistical-method selection, preprocessing completeness, package and runtime dependencies, random-state control, agreement between computed and reported values, figure specifications, and rerun stability are often assessed separately or omitted entirely. In addition, many current systems generate code and inspect the resulting output within the same conversational context, which may allow an agent to accept its own assumptions without independently verifying the complete workflow. Scientific analysis requires a stronger form of validation in which the workflow is reconstructed and rerun under controlled conditions, intermediate decisions are made explicit, and the final numerical and graphical outputs can be traced back to executable steps.

This study therefore presents SciFlow-Agent, a structured multi-stage framework for generating and verifying reproducible scientific data-analysis workflows. SciFlow-Agent separates data inspection, statistical planning, workflow implementation, visualization, and reproducibility verification so that analytical decisions can be evaluated at the stage where they are introduced rather than being hidden within a single code-generation step. The system examines dataset structure and variable properties before selecting an analytical procedure, converts the resulting statistical plan into executable Python or R workflows, generates figures according to task-specific reporting requirements, and subsequently reruns the complete workflow in controlled software environments. Dependency declarations, execution failures, numerical outputs,

intermediate result records, and consistency between computed values and reported conclusions are checked during verification. The framework is evaluated on 2,960 analysis tasks constructed from 740 public datasets spanning five scientific fields. Evaluation considers not only task completion but also statistical-method appropriateness, executable reproducibility, dependency completeness, numerical consistency, and conformity of figures and result records to the requested analysis. By treating scientific code generation as an auditable analytical workflow rather than a one-step programming problem, this study aims to establish a more rigorous evaluation framework for language-model-assisted research. The broader significance of SciFlow-Agent lies in reducing the gap between automatically generated code and reproducible scientific evidence, thereby supporting more transparent, scalable, and verifiable computational research across disciplines.

2. Materials and Methods

2.1. Dataset Sample and Study Scope

The benchmark contained 740 public datasets from biomedical research, environmental science, public health, economics, and social science. These five fields defined the study scope; no single geographic area was used. A dataset was included when its files were available, its variables were described, and its analysis aim was clear. Datasets were removed when files could not be opened, key variables were absent, or the required analysis could not be identified. The final benchmark included 2,960 analysis tasks, 8,420 expected statistical results, and 5,180 figure requirements. It covered cross-sectional, longitudinal, time-series, and grouped tabular data. Each task required data preparation, statistical analysis, and figure production.

2.2. Experimental Design and Control Setting

SciFlow-Agent was the experimental system. It used separate agents for data checking, method selection, code generation, figure generation, and reproducibility checking. The control system used one agent to generate code directly from the same task description. It had no separate planning, testing, or verification steps. Both systems used the same language model, prompt information, running time, and software environment. Tasks from the same dataset were placed in the same data split. This prevented similar data structures and reference results from appearing in both development and test sets. The two systems were tested on the same held-out tasks.

2.3. Measurement Methods and Quality Control

All Python and R scripts were run in clean container environments. A workflow passed only when it finished without manual changes and produced all required outputs. Generated statistical results were compared with reference values, including model estimates, test statistics, confidence intervals, and adjusted P values. Figures were checked for plot type, axis labels, legends, variable assignment, and required layout. Quality checks also covered package versions, input-file

checksums, run logs, and random seeds. Reference results were checked before the experiment. Tasks with unclear instructions were corrected or removed. A workflow was counted as reproducible only when it ran again in a new environment with the same data, listed packages, and matching results.

2.4. Data Processing and Evaluation Formulae

Data were processed in separate folders for each run. The system checked variable type, missing values, duplicate records, and unusual values before analysis. Original files were kept unchanged, and a cleaned copy was used for each run. Numerical outputs with different units were evaluated separately by output type. The numerical result-matching rate was calculated as

$$\text{RMR} = \frac{1}{N_{\text{num}}} \sum_{i=1}^{N_{\text{num}}} I(|\hat{y}_i - y_i^{\text{ref}}| \leq \tau_i)$$

where N_{num} is the number of numerical outputs, $\hat{y}_i$ is the result from the generated workflow, y_i^{ref} is the reference result, and τ_i is the preset tolerance for output i . $I(\cdot)$ equals 1 when the condition is met and 0 otherwise. The mean absolute error was calculated as

$$\text{MAE} = \frac{1}{N_{\text{num}}} \sum_{i=1}^{N_{\text{num}}} |\hat{y}_i - y_i^{\text{ref}}|$$

A lower MAE value indicated a smaller numerical difference from the reference output.

2.5. Reproducibility Verification and Statistical Analysis

Reproducibility was checked after temporary files were removed and the software environment was rebuilt from the dependency list. The verification agent checked required packages, file paths, analysis settings, output tables, and figures. The main outcomes were full-workflow execution rate and result-matching rate. Other outcomes included method-selection errors, package-related failures, mean absolute deviation, and figure accuracy. The two systems were compared on paired tasks. Binary outcomes were tested with a two-sided paired test, and confidence intervals were estimated by task resampling. Results were assessed at the 0.05 significance level.

3. Results and Discussion

3.1. Full-Workflow Execution Performance

SciFlow-Agent completed 79.5% of the 2,960 test tasks. The single-agent control completed 57.3%. The difference was 22.2 percentage points. A task was counted as successful only when data processing, statistical analysis, and figure generation were all completed in a clean environment. This standard was stricter than syntax checking or unit-test passing. SciFlow-Agent checked the data and selected a method before writing the final code. This order may reduce later errors. Fig. 1 shows a related sequence of environment setup, testing, and repair. DA-Code and SciCode assessed data-science or scientific code tasks [25,26]. In this study, each workflow also

had to rerun with recorded dependencies and produce the required numerical outputs and figures. The five agents were tested as one system, so the effect of each agent was not measured separately.

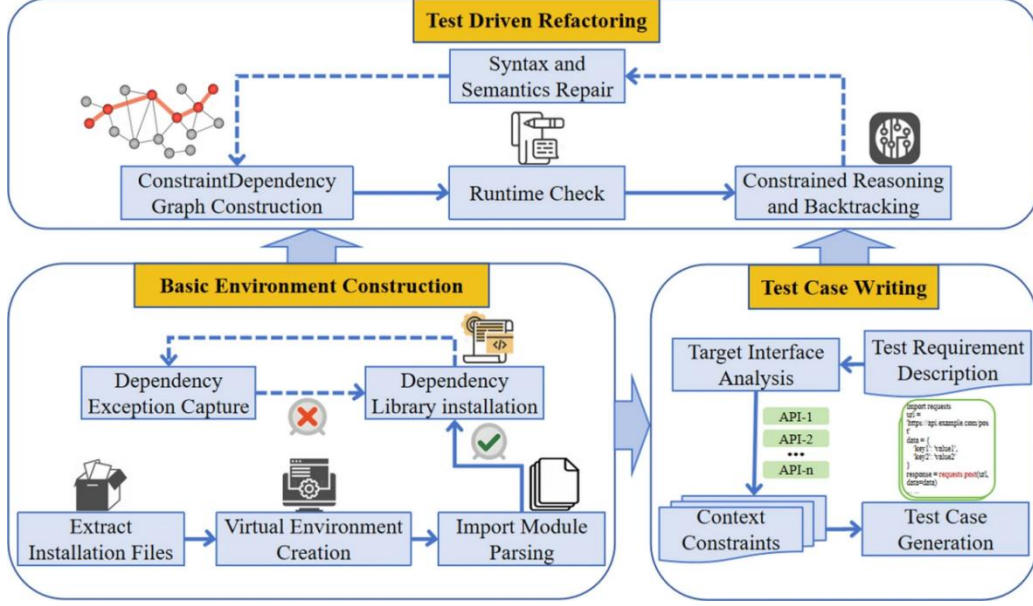


Fig. 1. Test-driven workflow from environment setup to code repair.

3.2. Reproducibility and Statistical Output Accuracy

The numerical result-matching rate increased from 61.8% to 84.2%. Statistical-method selection errors decreased by 36.7%, and dependency-related failures decreased by 48.6%. The MAE decreased from 0.084 to 0.031, which was a 63.1% reduction. These results show that SciFlow-Agent improved numerical agreement with the reference workflow. The output-level matching rate was higher than the full-workflow execution rate because it was calculated for individual numerical outputs. Full-workflow execution required every analysis step to finish. Earlier studies of shared research code and Jupyter notebooks found frequent problems with missing packages, unavailable files, and unclear running order [27,28]. The current system checked these issues during workflow execution. However, agreement with a reference output does not show that the selected statistical method is the only valid choice.

3.3. Figure Generation and Comparison with Existing Work

SciFlow-Agent met 86.9% of figure requirements, while the control system met 70.4%. The improvement was 16.5 percentage points. The figure checks covered plot type, variable mapping, axis labels, legends, and output format. An executable script may still produce an unsuitable figure. This check reduced such cases. The remaining error rate was 13.1%, which shows that figure generation still needs review when a task contains several variables or detailed journal requirements. Fig. 2 reports pass@n results from an automated code-evaluation study. Its values are not directly comparable with those in this study. AutoGEEval used geospatial unit-level

coding tasks, whereas SciFlow-Agent was tested on complete scientific workflows. Both studies used execution-based checks instead of text similarity alone.

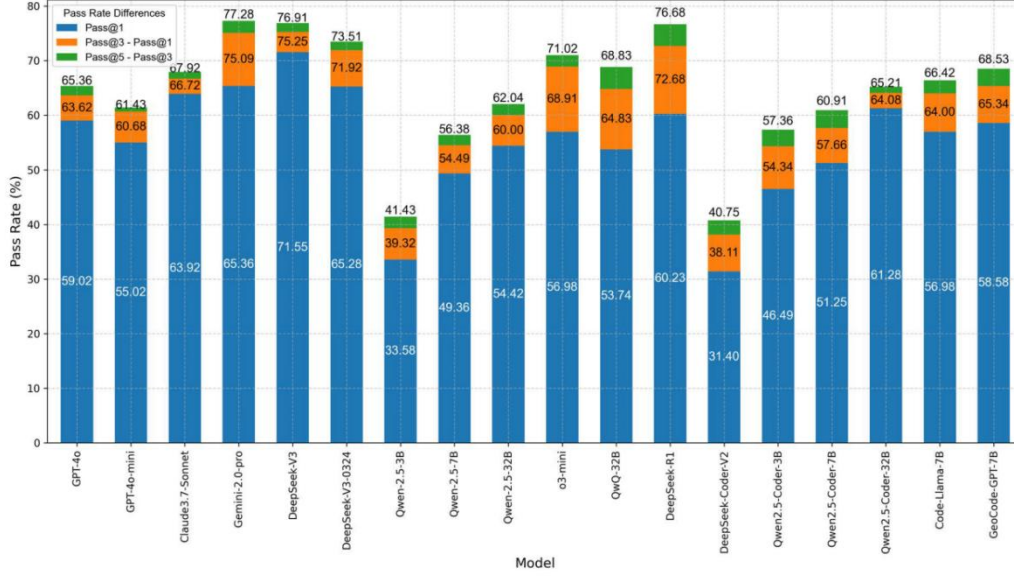


Fig. 2. Pass@n results for automated code evaluation.

3.4. Repair Efficiency and Study Limits

The mean number of repair cycles was 2.6. This result describes the repair cost under the benchmark setting. The datasets were public and had available documentation. Real projects may contain incomplete records, restricted data, or changing software packages. The benchmark tasks also had fixed reference workflows. In real research, more than one statistical method may be suitable for the same question. The system was tested only with Python and R. It did not assess data privacy, study design, or the scientific meaning of the final result. Later studies should test the system with external laboratory data, changing software environments, and expert review of statistical methods and figures [29,30].

4. Conclusion

This study examined SciFlow-Agent for scientific workflows that can be rerun. The system used five stages: data checking, statistical planning, code generation, figure generation, and reproducibility checking. Full-workflow completion increased from 57.3% to 79.5%, and numerical result matching increased from 61.8% to 84.2%. The mean absolute error decreased from 0.084 to 0.031. Figure-specification accuracy increased from 70.4% to 86.9%. The mean repair count was 2.6 cycles. Code that runs is not always a complete scientific workflow. The workflow also needs a suitable method, recorded packages, repeatable outputs, and a correct figure. SciFlow-Agent could help researchers prepare analysis scripts, share code, and check workflows before publication. However, the benchmark used public datasets, fixed reference workflows, and Python and R only. It did not test private data, open research questions, or the

scientific validity of all statistical choices. Future studies should test the system with laboratory data, changing software environments, and expert review.

References

1. Ahmad, R., Manne, N. N., & Malik, T. (2025). Improving reproducibility of interactive notebooks using application virtualization. *Future Generation Computer Systems*, 108043.
2. Zhao, J., Fan, J., & Li, L. (2026). A Study on an Explainable Causal-Enhanced LLM Agent for Predicting the Forming Quality of Automotive Component Materials.
3. You, S. (2026). Verifiable Audit Mechanisms in AI Compliance Automation: Scalability. Available at SSRN 6547458.
4. Yao, G., Zheng, J., Wang, Z., Zhang, W., Han, R., Zhao, C., ... & Liu, R. (2026, March). V-pruner: A fast and globally-informed token pruning framework for vision transformer. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 40, No. 40, pp. 34396-34404).
5. Collyer, T. A. (2026). Epistemic cultures, knowledge types, and disciplinary tensions in population health research: An international interview study. *Social Science & Medicine*, 119786.
6. Jiao, Y., Shi, T., Zhao, B., & Wang, A. (2026). Retrieval-Guided Structured Reasoning and Interpretable Representation Learning for Large-Scale Video–Language Models.
7. Liang, S., Du, Y., Chen, W., & Liu, Z. (2026). Order Allocation and Emergency Transportation Decisions Amid Multi-Source Procurement Disruptions.
8. Zhang, Z. (2026). A Study on the Impact of Cost Presentation on Decision-making Bias and Cancellation Behavior in Online Subscriptions. Available at SSRN 7349839.
9. Gülmez, B. (2026). Code generation with large language models: a survey from neural program synthesis to autonomous software development. *Applied Intelligence*, 56(6), 200.
10. Qi, C., & Qiao, X. (2026). Building and Operating a Large Scale Multi-Agent System: A Case Study from Industry. Available at SSRN 6795198.
11. Du, Y., Liu, Q., Dong, Y., & Chen, Z. (2026). Cross-Domain Transfer and Few-Shot Recognition of Defect Images in Complex Industrial Settings.
12. Naseem, R., & Zaki, M. (2026). Automated Code Generation Using Large Language Models: A Comparative Study. *Research Consortium Archive*, 4(2), 1178-1186.
13. Xiong, W., Guo, Y., & Zeng, Y. (2026). A Study on the Energy Efficiency of MEP Systems and Coordinated Dispatch Mechanisms for Multi-energy Systems in High-density Urban Buildings. Available at SSRN 7120518.
14. Liang, S., Zhang, X., Du, Y., & Chen, W. (2026). Supply Network Restructuring and Capacity Ramp-Up Limits in the Localized Expansion of High-Performance Computing Equipment Production. Available at SSRN 7339738.
15. Bao, Y., & Wang, H. (2026). Large-scale Metrics Migration: A Systematic Approach to Rebuilding Production Measurement Infrastructure. Available at SSRN 7185660.

16. Chopra, S., Basra, E. N. K., Simran, E., & Mohapatra, E. D. (Eds.). (2025). Fundamentals of Data Handling and Visualization. City, 100(15), 00-000.
17. Zhao, Z., & Welsch, R. E. (2024). Hierarchical reinforced trader (hrt): A bi-level approach for optimizing stock selection and execution. arXiv preprint arXiv:2410.14927.
18. Yuan, Y., Xu, T., Yin, J., & Huang, J. (2026). Modeling Loading Anomalies and Identifying Root Causes in Media Consumption Workflows on Large Social Media Platforms. Available at SSRN 7393318.
19. Bao, Y., & Wang, H. (2026). Orion: A High-Throughput, Fault-Tolerant Event Routing Architecture for Hyperscale Data Streams. Fault-Tolerant Event Routing Architecture for Hyperscale Data Streams (January 01, 2026).
20. Otoum, N., & Elkhaili, N. (2026). Methods and techniques of agentic software engineering: A systematic literature review. IEEE Access, 14, 7443-7465.
21. Wang, S., Feng, Y., & Fang, X. (2026, May). A Large Language Model-Enabled Multi-Agent Collaboration Method for Complex Task Solving. In 2026 6th International Symposium on Computer Technology and Information Science (ISCTIS) (pp. 253-256). IEEE.
22. Yang, Q., & Du, Y. (2026). Predicting Rollback Risks and Controlling Gradual Rollout Traffic for Online Ranking Feature Deployments.
23. Ferdaus, M. M., Abdelguerfi, M., Loup, E., N. Niles, K., Pathak, K., & Sloan, S. (2026). Towards trustworthy ai: A review of ethical and robust large language models. ACM Computing Surveys, 58(7), 1-43.
24. Yuan, Y., Yin, J., Huang, J., & Xu, T. (2026). Weakly Supervised Anomaly Detection and Privacy Risk Scoring in Community Message Threads.
25. Hong, Z., Liang, S., Xu, T., & Chen, H. (2026). A Study on Failover Verification and Recovery Objective Prediction for Cross-Region Cloud Services.
26. Trofimova, E., Shamina, Z., Selifanova, M., Zaitsev, A., Savchuk, R., Minets, M., ... & Ustyuzhanin, A. E. (2025). ML2B: Multi-Lingual ML Benchmark For AutoML. arXiv preprint arXiv:2509.22768.
27. Wu, J., Wu, D., Zhang, J., & Peng, Y. (2026). Generative AI Feedback in Junior High School Artistic Creation Evaluation. Available at SSRN 7244838.
28. Huang, J., Xu, T., Yin, J., & Yang, J. (2026). Data Quality Monitoring and Intelligent Scheduling Optimization in Financial Data Workflow Automation. Available at SSRN 7179338.
29. Souaifi, M., Dhahbi, W., Jebabli, N., Ceylan, H. İ., Boujabli, M., Muntean, R. I., & Dergaa, I. (2025). Artificial intelligence in sports biomechanics: A scoping review on wearable technology, motion analysis, and injury prevention. Bioengineering, 12(8), 887.
30. Zhang, Z., Wang, J., Li, Z., Wang, Y., & Zheng, J. (2025). Annocoder: A mti-agent-based code generation and optimization model. Symmetry, 17(7), 1087.