

# Rxiv-Maker: an automated template engine for streamlined scientific publications

Bruno M. Saraiva<sup>1,✉</sup>, António D. Brito<sup>1</sup>, Guillaume Jaquemet<sup>2,3,4,✉</sup>, and Ricardo Henriques<sup>1,5,✉</sup>

<sup>1</sup>Instituto de Tecnologia Química e Biológica António Xavier, Universidade Nova de Lisboa, Oeiras, Portugal

<sup>2</sup>Faculty of Science and Engineering, Cell Biology, Åbo Akademi University, Turku, Finland

<sup>3</sup>InFLAMES Research Flagship Center, University of Turku, Turku, Finland

<sup>4</sup>Turku Bioscience Centre, University of Turku and Åbo Akademi University, Turku, Finland

<sup>5</sup>UCL Laboratory for Molecular Cell Biology, University College London, London, United Kingdom

The rapid growth of preprint servers has accelerated scientific dissemination but has also shifted the technical burden of manuscript preparation to authors. This challenge is particularly acute in computational research, where manuscripts must remain synchronised with evolving data and code. We present Rxiv-Maker, a framework that resolves this by converting simple Markdown files into professionally typeset, publication-ready PDFs. Its core feature is the ability to execute embedded code, creating a self-updating manuscript where figures and statistical values are generated directly from source data during compilation. This ensures that the final document is always current and fully reproducible. By integrating with standard tools like Git and Visual Studio (VS) Code, Rxiv-Maker provides an efficient, transparent, and collaborative authoring experience, applying principles of software engineering to academic writing to foster open and verifiable science.

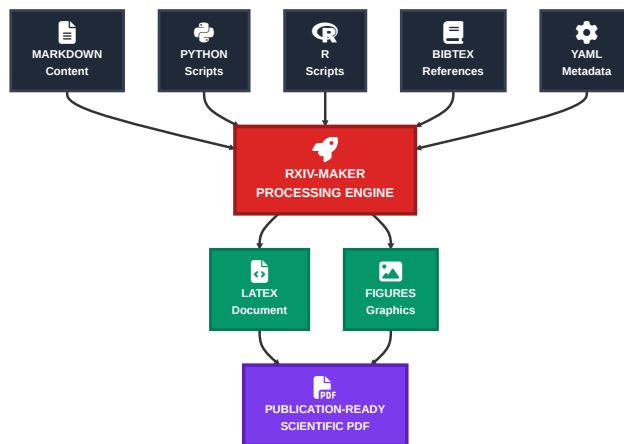
article template | scientific publishing | preprints

Correspondence: (B. M. Saraiva) [bsaraiva@itqb.unl.pt](mailto:bsaraiva@itqb.unl.pt); (G. Jaquemet) [guillaume.jacquemet@abo.fi](mailto:guillaume.jacquemet@abo.fi); (R. Henriques) [ricardo.henriques@itqb.unl.pt](mailto:ricardo.henriques@itqb.unl.pt)

## Introduction

The landscape of scientific publishing has been profoundly reshaped by the rise of preprint servers, which facilitate rapid dissemination of research findings across numerous platforms (1–3) (Fig. S1, Fig. S2). This acceleration, however, has transferred the complex task of typesetting from publishers to researchers (4–6). For those of us in computational fields, this is compounded by a more pressing challenge: ensuring the manuscript remains perfectly synchronised with our data and analysis code. We have all faced the tedious and error-prone task of manually updating a p-value or a sample size in the text after re-running an analysis (7). This disconnect between the research and the report is a critical issue, particularly in disciplines like bioimage analysis, where our findings are built upon complex computational pipelines (8, 9).

To address these pain points, we developed Rxiv-Maker, a framework designed to streamline the creation of scientific manuscripts. It automates the production of publication-ready PDFs from simple Markdown, harnessing the typesetting power of LaTeX without requiring the author to write LaTeX code. Rxiv-Maker transforms the document into a self-updating manuscript by executing embedded Python or R scripts during compilation. This approach forges a direct, traceable link between the data and the final publication, eliminating manual transcription errors and ensuring the document is always synchronised with the latest results. The frame-

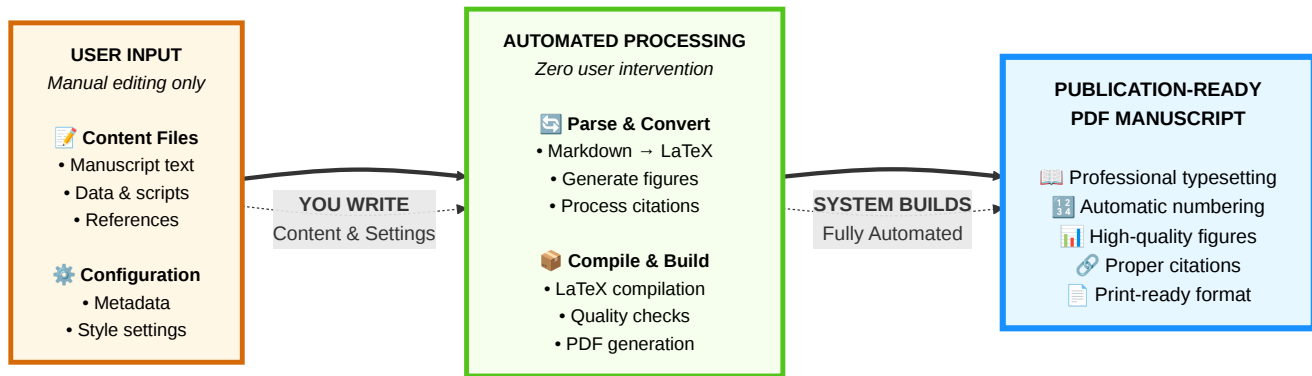


**Fig. 1. System Architecture.** Rxiv-Maker integrates Markdown content, YAML metadata, executable scripts, and bibliographies through a processing engine that combines local execution with LaTeX compilation to produce publication-ready PDFs.

work is designed as a local-first system, providing a rapid and responsive authoring experience that integrates seamlessly with version control systems like Git, thereby enhancing transparency and collaboration (10, 11). The overall architecture and processing pipeline are illustrated in Fig. 1 and Fig. 2, respectively. A tutorial is provided to guide new users through their first manuscript (Supp. Note 1).

## Results

**From Static Text to Self-Updating Manuscripts.** A central achievement of Rxiv-Maker is its ability to create dynamic, self-updating documents. By embedding executable code snippets directly within the Markdown source, the manuscript transitions from a static report to a living document. This functionality enables the direct execution of analysis scripts during compilation, with the results (whether statistical values, tables, or figures) directly injected into the text. This process eliminates the possibility of transcription errors when, for example, copying a value from a terminal or another program into the manuscript. We demonstrate this with a practical example from this very paper. The Python code shown in Fig. 3 is embedded in the source of our manuscript. At build time, it retrieves the latest public data on arXiv submissions, calculates key statistics, and inserts them into the text. This ensures that our discussion of preprint growth is always based on the most current evidence available. The analysis reveals that, since 1991, a total of 2.84 million submissions have been



**Fig. 2. Processing Pipeline.** User-provided content (left) undergoes automated processing (right), including parsing, script execution, LaTeX conversion, and PDF compilation.

made over 35 years (Fig. S1). These are not static numbers; they were computed when this document was compiled on September 29, 2025, using data updated in September 2025. This capability extends to all forms of programmatic figure generation, as detailed in [Supp. Note 3](#), ensuring that every visual and numerical claim is reproducibly generated.

**Automated Professional Typesetting from Simple Text.** Rxiv-Maker allows researchers to achieve the typographic quality of LaTeX while writing in simple, intuitive Markdown. This is accomplished through a sophisticated, multi-pass translator that converts Markdown to LaTeX in a guarded manner. We recognised that a naive, single-pass conversion would be too fragile for academic documents, where complex syntax for mathematics, citations, and cross-references must be preserved. Our translator first identifies and protects delicate elements: mathematical expressions ([Supp. Note 4](#)), code blocks, and citation keys. Subsequent passes then normalise document structure and intelligently convert Rxiv-Maker’s ex-

tended Markdown syntax into the corresponding LaTeX commands, supporting comprehensive text formatting including **bold**, *italic*, underlined, and other typographic elements. The system handles complex nested formatting combinations such as **bold within underline** and **underlined within bold** seamlessly. For instance, it transforms image syntax into floating figure environments, automatically handling captions, labels, and layout parameters. This process enables authors to specify figure widths or positions using simple options in Markdown, without needing to modify LaTeX code. The system supports a wide array of figure generation methods, from static images to script-based visualisations (Table S1), ensuring all visual elements are correctly and professionally typeset. For more advanced formatting needs, including complex tables, users can directly inject LaTeX code using `{\tex: ...}` blocks. A complete syntax reference and example are detailed in [Supp. Note 2](#).

**An Integrated and Rapid Authoring Experience.** We designed Rxiv-Maker to provide a seamless and efficient writing experience. The framework operates as a local-first command-line tool, which means that compilation is fast and authors receive immediate feedback on their changes. This rapid iteration is enhanced by an intelligent caching system that remembers which figures and results are already built. It computes a unique signature: a checksum from a figure’s source code and data dependencies. A script is only re-executed if this signature has changed, saving valuable time during compilation without compromising reproducibility, as further explained in [Supp. Note 5](#). To lower the barrier to entry, we developed a companion extension for Visual Studio Code (Fig. S3). This extension acts as a helpful assistant, providing syntax highlighting for our extended Markdown, autocompletion for citation keys and cross-references, and real-time validation of the manuscript’s configuration. This integration of authoring and validation into a single, familiar environment streamlines the writing process, allowing researchers to focus on their content rather than on the technical details of typesetting.

**Designed for Transparent and Collaborative Science.** Reproducibility and collaboration are at the heart of Rxiv-Maker. By design, the entire manuscript is structured to be managed with Git, the version control system that is standard in soft-

```

{{py:exec
import pandas as pd
from datetime import datetime
from pathlib import Path
from data_updater import update_all_data_files

# Update data files and load arXiv statistics
update_all_data_files()
df = pd.read_csv(Path("DATA") / "arxiv_monthly_submissions.csv")
df['year_month'] = pd.to_datetime(df['year_month'])

# Calculate key statistics for the manuscript
data_start_year = int(df['year_month'].dt.year.min())
total_submissions = int(df['submissions'].sum())
total_submissions_millions = round(total_submissions / 1_000_000, 2)
years_span = int(df['year_month'].dt.year.max() - df['year_month'].dt.
year.min() + 1)
compilation_date = datetime.now().strftime("%B %d, %Y")
last_updated = datetime.fromtimestamp(Path("DATA") /
"arxiv_monthly_submissions.csv").stat().st_mtime).strftime("%B %Y")

print(f"Loaded {len(df)} months of arXiv data spanning {years_span}
years")
}}

```

**Fig. 3. Embedded Python for dynamic content.** This figure shows a Python script embedded in the manuscript’s source code. The script is executed at build time to compute data attributes (e.g., total submissions and span of years) and inject the resulting values directly into the text. This process, rendered here with syntax highlighting from the Rxiv-Maker VS Code extension, eliminates manual transcription errors and ensures the text remains synchronised with the source data.

ware development. This includes the text, code, data, and configuration. This treats the manuscript as a complete, self-contained project where every change is tracked, attributed, and auditable. This aligns perfectly with the principles of open science, creating a transparent history of the research from its inception to the final publication. The framework's command-line interface includes a powerful `rxiv validate` command that serves as a quality control gatekeeper. It performs a series of checks, such as verifying that all figures are present, all cross-references are valid, and all bibliographic entries are correctly formatted. This validation can be integrated into automated workflows, for instance, as a pre-commit hook to prevent broken versions from entering the project history, or as a check in a continuous integration (CI) pipeline to ensure that a manuscript is always ready for dissemination. This brings the rigour of professional software engineering to the academic writing process.

## Discussion

Rxiv-Maker finds its place in a growing ecosystem of tools designed to enhance scientific publishing. Our approach is distinct from collaborative web-based editors like Overleaf (12), which, while excellent for team-based LaTeX writing, do not address the fundamental challenge of keeping a manuscript synchronised with external data and analysis code (13). Our philosophy is more closely aligned with that of platforms like Manubot (14), Jupyter Book (15), and Quarto (16), which also integrate code and narrative, often building on notebook-based formats (17). However, Rxiv-Maker is specifically optimised for the production of high-quality, submission-ready manuscripts from a local-first, developer-centric workflow. Beyond preprint distribution, rxiv-maker excels at generating professional manuscripts suitable for first submission to peer-reviewed journals, providing publication-quality PDFs and clean LaTeX source code that many journals readily accept prior to their final journal-specific formatting requirements. This positions rxiv-maker as a comprehensive solution spanning the entire academic publishing pipeline—from initial draft through preprint distribution to journal submission. As shown in our comparison with other tools (Table S2), its unique strengths lie in its self-updating capabilities, its intelligent caching for rapid rebuilds, and its deep integration with the Git-based workflows that are common in computational research. We also note other modern typesetting systems like Typst (18) and frameworks like Bookdown (19), which offer alternative paradigms for scientific document creation. While powerful, Rxiv-Maker has limitations. Its primary output is currently PDF, which is ideal for preprints but less suited for the web. To address this, we plan to integrate Pandoc (20) into our pipeline, which will enable multi-format outputs, such as HTML and EPUB, while preserving our commitment to typographic quality. We also intend to enhance the framework's integration with computational environment managers to further improve the portability and reproducibility of our executable manuscripts across different systems and institutions (8, 9), for which containerisation technologies are also a valid alternative (21, 22). By applying the principles of literate

programming (23) to the creation of academic papers, Rxiv-Maker helps bridge the gap between computational research and traditional scientific publishing. It fosters a workflow where the manuscript is not merely a report of the research, but a reproducible and verifiable embodiment of it. In an era where the responsibility for typesetting and validation increasingly falls to authors, tools that automate quality control and promote rigour are essential. Rxiv-Maker is our contribution to this movement, offering a framework that empowers researchers to produce transparent, reproducible, and professionally polished manuscripts with confidence and efficiency.

### ABOUT THIS MANUSCRIPT

This work is licensed under CC BY 4.0.

### DATA AVAILABILITY

The arXiv monthly submission data used in this article is available at [https://arxiv.org/stats/monthly\\_submissions](https://arxiv.org/stats/monthly_submissions). Preprint submissions data across different hosting platforms is available at <https://github.com/esperr/pubmed-by-year>. The source code and data for the figures in this article are available at <https://github.com/HenriquesLab/rxiv-maker>.

### CODE AVAILABILITY

The Rxiv-Maker computational framework is available at <https://github.com/HenriquesLab/rxiv-maker>. The companion Visual Studio Code extension is at <https://github.com/HenriquesLab/vscode-rxiv-maker>. For users requiring containerised execution, the docker-rxiv-maker repository provides Docker-based deployment at <https://github.com/HenriquesLab/docker-rxiv-maker>. All repositories are released under an MIT License.












### AUTHOR CONTRIBUTIONS

Bruno M. Saraiva, Guillaume Jacquemet, and Ricardo Henriques conceived the project and designed the framework. António D. Brito beta-tested features and troubleshooted user experience. All authors contributed to writing and reviewing the manuscript.

### ACKNOWLEDGEMENTS

The authors thank Jeffrey Perkel for feedback that helped improve the manuscript. B.S. and R.H. acknowledge support from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 101001332) (to R.H.) and funding from the European Union through the Horizon Europe program (AI4LIFE project with grant agreement 101057970-AI4LIFE and RT-SuperES project with grant agreement 101099654-RTSuperES to R.H.). Funded by the European Union. However, the views and opinions expressed are those of the authors only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them. This work was also supported by a European Molecular Biology Organization (EMBO) installation grant (EMBO-2020-IG-4734 to R.H.), a Chan Zuckerberg Initiative Visual Proteomics Grant (vpi-0000000044 with <https://doi.org/10.37921/743590vtdfp> to R.H.), and a Chan Zuckerberg Initiative Essential Open Source Software for Science (EOSS6-0000000260). This study was supported by the Academy of Finland (no. 338537 to G.J.), the Sigrid Juselius Foundation (to G.J.), the Cancer Society of Finland (Syöpäjärjestöt, to G.J.), and the Solutions for Health strategic funding to Åbo Akademi University (to G.J.). This research was supported by the InFLAMES Flagship Program of the Academy of Finland (decision no. 337531).

### EXTENDED AUTHOR INFORMATION

- **Bruno M. Saraiva:**  
 0000-0002-9151-5477;  Bruno\_MSaraiva;  bruno-saraiva
- **António D. Brito:**  
 0009-0001-1769-2627
- **Guillaume Jacquemet:**  
 0000-0002-9286-920X;  guijacquemet;  guijacquemet.bsky.social
- **Ricardo Henriques:**  
 0000-0002-2043-5234;  HenriquesLab;  henriqueslab.bsky.social;  
 ricardo-henriques

## Bibliography

1. Jeffrey Beck, Christine A Ferguson, Kathryn Funk, Brooks Hanson, Melissa Harrison, Michele Ide-Smith, Rachael Lammey, Maria Levchenko, Alex Mendonca, Michael Parkin, Naomi Penfold, Nicole Pfeiffer, Jessica Polka, Iratxe Puebla, Oya Y Rieger, Martyn Rittman, Richard Sever, and Sowmya Swaminathan. Building trust in preprints: recommendations for servers and other stakeholders, 2020.
2. Mariia Levchenko, Michael Parkin, Johanna McEntyre, and Melissa Harrison. Enabling preprint discovery, evaluation, and analysis with europe pmc, 2024.

3. Nicholas Fraser, Fakhri Momeni, Philipp Mayr, and Isabella Peters. The relationship between biorxiv preprints, citations and altmetrics. *Quantitative Science Studies*, 1(2):618–638, 2020. doi: 10.1162/qss\_a\_00043.
4. Ronald D Vale. Accelerating scientific publication in biology. *Proceedings of the National Academy of Sciences*, 112(44):13439–13446, 2015. doi: 10.1073/pnas.1511912112.
5. Jonathan P Tenant, Francois Waldner, Damien C Jacques, Paola Masuzzo, Lauren B Collister, and Chris HJ Hartgerink. The academic, economic and societal impacts of open access: an evidence-based review. *F1000Research*, 5:632, 2016. doi: 10.12688/f1000research.8460.3.
6. Jialiang Lin, Yao Yu, Yu Zhou, Zhiyang Zhou, and Xiaodong Shi. How many preprints have actually been printed and why: a case study of computer science preprints on arxiv. *Scientometrics*, 124(1):555–574, 2020. doi: 10.1007/s11192-020-03430-8.
7. Jeffrey M. Perkel. Cut the tyranny of copy-and-paste with these coding tools. *Nature*, 603(7899):191–192, 2022. doi: 10.1038/d41586-022-00563-z.
8. Ulysse Rubens, Romain Mormont, Lassi Paavolainen, Volker Bäcker, Benjamin Pavie, et al. Biflows: A collaborative framework to reproducibly deploy and benchmark bioimage analysis workflows. *Patterns*, 1(3):100040, 2020. doi: 10.1016/j.patter.2020.100040.
9. Ivan Hidalgo-Cenamor, Joanna W Pylvänäinen, Mariana G Ferreira, et al. D4miceverywhere: deep learning for microscopy made flexible, shareable and reproducible. *Nature Methods*, 21(9):1645–1656, 2024. doi: 10.1038/s41592-024-02295-6.
10. Karthik Ram. Git can facilitate greater reproducibility and increased transparency in science. *Source Code for Biology and Medicine*, 8(1):7, 2013. doi: 10.1186/1751-0473-8-7.
11. Yasset Perez-Riverol, Laurent Gatto, Rui Wang, Timo Sachsenberg, Julian Uszkoreit, Felipe da Veiga Leprevost, Christian Fufezan, Tobias Ternent, Stephen J Eglen, Daniel S Katz, et al. Ten simple rules for taking advantage of git and github. *PLoS Computational Biology*, 12(7):e1004947, 2016. doi: 10.1371/journal.pcbi.1004947.
12. Overleaf. Overleaf: Real-time collaborative writing and publishing tools with integrated pdf preview, 2024. Cloud-based LaTeX editor.
13. Ricardo Henriques. Henriques biorxiv template, 2015. Overleaf LaTeX template. Accessed: 2025-06-16.
14. Daniel S. Himmelstein, Vincent Rubinetti, David R. Siochower, Dongbo Hu, Venkat S. Malladi, Casey S. Greene, and Anthony Gitter. Open collaborative writing with manubot. *PLoS Computational Biology*, 15(6):e1007128, 2019. doi: 10.1371/journal.pcbi.1007128.
15. The Executable Book Community. Jupyter book: Build beautiful, publication-quality books and documents from computational material, 2020. Accessed: 2025-09-24.
16. Posit PBC. Quarto: An open-source scientific and technical publishing system, 2024. Multi-language scientific publishing system.
17. Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. Jupyter notebooks—a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90. IOS Press, 2016. doi: 10.3233/978-1-61499-649-1-87.
18. Typst GmbH. Typst: A new markup-based typesetting system, 2024. Modern typesetting system designed for scientific documents.
19. Yihui Xie. *bookdown: Authoring Books and Technical Documents with R Markdown*. Chapman and Hall/CRC, 2016. ISBN 9781138700109.
20. Pandoc: The universal markup converter, 2020. Accessed: 2025-06-16.
21. Carl Boettiger. An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, 2015. doi: 10.1145/2723872.2723882.
22. Jorge Gomes, Emanuele Bagnaschi, Isabel Campos, Mario David, Luis Alves, João Martins, João Pina, Alvaro López-García, and Pablo Orviz. Enabling rootless linux containers in multi-user environments: The udocker tool. *Computer Physics Communications*, 232:84–97, 2018. doi: 10.1016/j.cpc.2018.05.021.
23. Donald E Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984. doi: 10.1093/comjnl/27.2.97.
24. John D Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.
25. Michael L Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(6):3021, 2021. doi: 10.21105/joss.03021.
26. Mermaid Team. Mermaid: Generation of diagrams and flowcharts from text in a similar manner as markdown, 2023. Accessed: 2024-12-01.
27. Sperr, Ed. Pubmed by year: A dataset of pubmed publication counts by year, 2025. Accessed: 2025-07-01.

## Methods

The Rxiv-Maker framework is implemented as a modular, Python-based engine designed to ensure computational reproducibility at every stage of manuscript preparation. The processing pipeline begins by establishing an isolated and consistent computational environment, where all dependencies for Python, R, and LaTeX are managed. This ensures that a manuscript can be reliably recompiled on different machines and in the future. The framework’s overall architecture is detailed in Fig. S4. The core of our approach is transforming the manuscript into an executable document. The engine identifies and runs embedded Python or R scripts to generate figures and statistical values directly from source data, as described in Supp. Note 3. To ensure a rapid and

interactive authoring experience, this process is optimised with an intelligent caching system. It computes content-based checksums of the source scripts and their data dependencies, so that code is only re-executed when a genuine change has been made (Supp. Note 5). This saves significant time during iterative writing without compromising the integrity of the results. To convert the author’s simple Markdown into a professional LaTeX document, we developed a robust, multi-pass translator. This translator employs a content-protection strategy, first identifying and isolating delicate syntactic structures, including mathematical equations (Supp. Note 4), citation keys, and code blocks. Subsequent passes then normalise the document structure and map Rxiv-Maker’s extended syntax for figures and cross-references to the appropriate LaTeX commands. This guarded approach enables authors to leverage the simplicity of Markdown without compromising the complex syntax necessary for academic publishing. A multi-level validation system provides continuous quality control. Before compilation, it checks for common errors, such as missing files or broken references. During compilation, it parses LaTeX and BibTeX logs to provide clear, actionable error messages. This automated quality control is essential for maintaining the integrity of the document throughout the authoring process. Finally, to ensure maximum reproducibility, the framework supports multiple deployment strategies. The primary recommended approach is a local installation, which requires Python and a LaTeX distribution. This method offers an excellent combination of high usability and reproducibility, with collaboration managed through Git. For absolute environmental consistency, which is particularly valuable for large collaborations or long-term archival, a fully containerised version is also available. This requires Docker but encapsulates the entire build environment, guaranteeing that the manuscript can be recompiled with bit-for-bit identical results on any system.

## Supplementary Information

**R $\chi$ iv-Maker: an automated template engine for  
streamlined scientific publications**



Format	Input Extension	Processing Method	Output Formats	Quality	Use Case
Mermaid Diagrams	.mmd	Mermaid CLI	SVG, PNG, PDF	Vector/Raster	Flowcharts, architectures
Python and R Figures	.py, .R	Script execution	PNG, PDF, SVG	Publication	Data visualisation
Static Images	.png, .jpg, .svg	Direct inclusion	Same format	Original	Photographs, logos
LaTeX Graphics	.tex, .tikz	LaTeX compilation	PDF	Vector	Mathematical diagrams
Data Files	.csv, .json, .xlsx	Python and R processing	Via scripts	Computed	Raw data integration

**Sup. Table S1. Supported Figure Generation Methods.** Overview of the framework's figure processing capabilities, demonstrating support for both static and dynamic content generation with an emphasis on reproducible computational graphics.

Tool	Type	Markdown	Primary Use Case	Key Strengths	Open Source
Rxiv-Maker	Pipeline	Excellent	Reproducible preprints	Local-first execution, automated caching, rich CLI	Yes
Overleaf (12)	Web Editor	Limited	Collaborative LaTeX	Real-time collaboration, rich templates, cloud-based	Freemium
Quarto (16)	Publisher	Native	Multi-format publishing	Polyglot support, multiple outputs, scientific focus	Yes
Manubot (14)	Collaborative	Native	Version-controlled writing	Automated citations, transparent collaboration, Git-based	Yes
Pandoc (20)	Converter	Excellent	Format conversion	Universal format support, extensible filters	Yes
Typst (18)	Typesetter	Good	Modern typesetting	Fast compilation, modern syntax, growing ecosystem	Yes
Bookdown (19)	Publisher	R Markdown	Academic books	Cross-references, multiple formats	Yes
Direct LaTeX	Typesetter	None	Traditional publishing	Full control, established workflows, mature ecosystem	Yes

**Sup. Table S2. Comparison of Manuscript Preparation Tools.** This comparison positions each tool within the scientific publishing ecosystem. Rxiv-Maker specialises in reproducible preprint workflows with local-first execution and developer-centric features. Other tools address distinct needs, such as real-time collaborative editing (Overleaf), multi-format output (Quarto), or version-controlled writing (Manubot).

**Supp. Note 1: A Getting Started Tutorial.** To help new users begin with Rxiv-Maker, this tutorial provides a simple walkthrough for creating a minimal manuscript. The process starts with initialising a new project, which sets up the necessary directory structure and configuration files.

First, open a terminal and run the command `rxiv init my_new_paper`. This creates a new directory named `my_new_paper` containing the essential files: `00_CONFIG.yml` for manuscript metadata, `01_MAIN.md` for the main text, `02_SUPPLEMENTARY_INFO.md` for supplementary content, and `03_REFERENCES.bib` for the bibliography. The system also creates subdirectories for `FIGURES` and `DATA`.

Next, you can edit the `01_MAIN.md` file to add your text. You can use standard Markdown for formatting, such as `##` for headings and `*` for italics. To add a citation, simply use the `@key` syntax, where `key` corresponds to an entry in your `.bib` file. For example, to cite the original literate programming paper, you would write `[@Knuth1984_literate_programming]`. To build the PDF, navigate into the project directory (`cd my_new_paper`) and run the command `rxiv pdf`. The framework will process your files, execute any embedded code, and generate a professionally typeset PDF in the `output` directory. If you encounter any issues, running `rxiv validate` provides a detailed check of your manuscript’s integrity, flagging common problems like missing figures or broken citations.

This straightforward process allows you to get from a blank slate to a compiled PDF in minutes, providing a solid foundation that you can then build upon with more advanced features like programmatic figures and tables.

**Supp. Note 2: Rxiv-Maker Markdown Syntax and Advanced LaTeX Integration.** This comprehensive reference demonstrates the automated translation system that enables researchers to write in familiar markdown syntax whilst producing professional LaTeX output. The table below showcases both the standard Markdown-to-LaTeX translations and serves as a perfect example of Rxiv-Maker’s `{\tex: . . . }` blocks, which allow direct LaTeX injection for advanced formatting needs. For complex table structures that require precise control over formatting, multi-column headers, or mathematical notation, Rxiv-Maker’s `{\tex: . . . }` syntax provides full access to LaTeX’s typesetting capabilities. This table itself was created using `{\tex: . . . }` blocks, demonstrating how raw LaTeX can be seamlessly integrated into Markdown documents while maintaining the tex block protection system that prevents markdown processing of LaTeX-specific syntax.

Markdown Input	LaTeX Output	Description
Basic Text Formatting		
<code>**bold text**</code>	<code>\textbf{bold text}</code>	Bold formatting
<code>*italic text*</code>	<code>\textit{italic text}</code>	Italic formatting
<code>__underlined text__</code>	<code>\underline{underlined text}</code>	Underlined formatting for emphasis
<code>** __bold and underlined__ **</code>	<code>\textbf{\underline{bold and underlined}}</code>	Nested formatting: bold containing underline
<code>__**underlined and bold**__</code>	<code>\underline{\textbf{underlined and bold}}</code>	Nested formatting: underline containing bold
<code>* __italic and underlined__ *</code>	<code>\textit{\underline{italic and underlined}}</code>	Multiple formatting combinations
<code>~subscript~</code>	<code>\textsubscript{subscript}</code>	Subscript formatting, e.g., H <sub>2</sub> O, CO <sub>2</sub>
<code>^superscript^</code>	<code>\textsuperscript{superscript}</code>	Superscript formatting, e.g., E=mc <sup>2</sup> , x <sup>n</sup>
Document Structure		
<code># Header 1</code>	<code>\section{Header 1}</code>	Top-level section
<code>## Header 2</code>	<code>\subsection{Header 2}</code>	Second-level section
<code>### Header 3</code>	<code>\subsubsection{Header 3}</code>	Third-level section
Lists		
<code>- list item</code>	<code>\begin{itemize}\item...\end{itemize}</code>	Unordered list
<code>1. list item</code>	<code>\begin{enumerate}\item...\end{enumerate}</code>	Ordered list
Links and URLs		
<code>[link text](url)</code>	<code>\href{url}{link text}</code>	Hyperlink with custom text
<code>https://example.com</code>	<code>\url{https://example.com}</code>	Bare URL
Citations		
<code>@mycitation</code>	<code>\cite{mycitation}</code>	Single citation
<code>[@himmelstein2019;@Overleaf2024]</code>	<code>\cite{himmelstein2019,Overleaf2024}</code>	Multiple citations
Cross-References		
<code>figure:label</code>	<code>\ref{fig:label}</code>	Figure reference
<code>supfig:label</code>	<code>\ref{sfig:label}</code>	Supplementary figure
<code>table:label</code>	<code>\ref{table:label}</code>	Table reference
<code>suptab:label</code>	<code>\ref{stable:label}</code>	Supplementary table
<code>equation:label</code>	<code>\eqref{eq:label}</code>	Equation reference
<code>note:label</code>	<code>\sidenote{label}</code>	Supplement note reference
Tables and Figures		
<code>Markdown table</code>	<code>\begin{table}...\end{table}</code>	Automatic table formatting
<code>Image with caption</code>	<code>\begin{figure}...\end{figure}</code>	Figure with caption

Markdown Input	LaTeX Output	Description
<b>Document Control</b>		
<!-- comment -->	% comment	LaTeX comments
<newpage>	\newpage	Manual page break
<clearpage>	\clearpage	Page break with float clearing

**Sup. Table 3. Rxiv-Maker Markdown to LaTeX Translation Reference.** Comprehensive mapping of Markdown syntax to corresponding LaTeX commands, demonstrating the automated translation system that enables researchers to write in familiar markup while producing professional typesetting.

**Supp. Note 3: Programmatic Figure Generation.** Rxiv-Maker’s figure generation capabilities are designed to ensure a transparent and reproducible connection between your data and your final visualisations. The system supports two main approaches for creating figures programmatically: script-based generation using Python or R, and diagram rendering from text-based descriptions using Mermaid.

For script-based figures, you place your `.py` or `.R` scripts in the `FIGURES` directory. These scripts often leverage powerful plotting libraries such as Matplotlib (24) or Seaborn (25). During compilation, Rxiv-Maker executes these scripts, and any image files they save (e.g., PNG, PDF, SVG) are automatically detected and can be included in your manuscript. This ensures your visualisations are always synchronised with the underlying data and analysis, as a change in one will trigger the regeneration of the other. This is the method used to produce Fig. S1 and Fig. S2.

For diagrams, such as flowcharts or system architectures, you can use Mermaid (26). You create a `.mmd` file containing a text-based description of your diagram. The framework uses the Mermaid command-line tool to render this description into a vector or raster image. This allows your diagrams to be version-controlled just like code, making them easy to modify and track over time.

**Supp. Note 4: Mathematical Formula Support.** Rxiv-Maker seamlessly integrates mathematical notation by translating Markdown-style expressions into high-quality LaTeX mathematics. This allows you to write complex mathematical content using simple, familiar syntax.

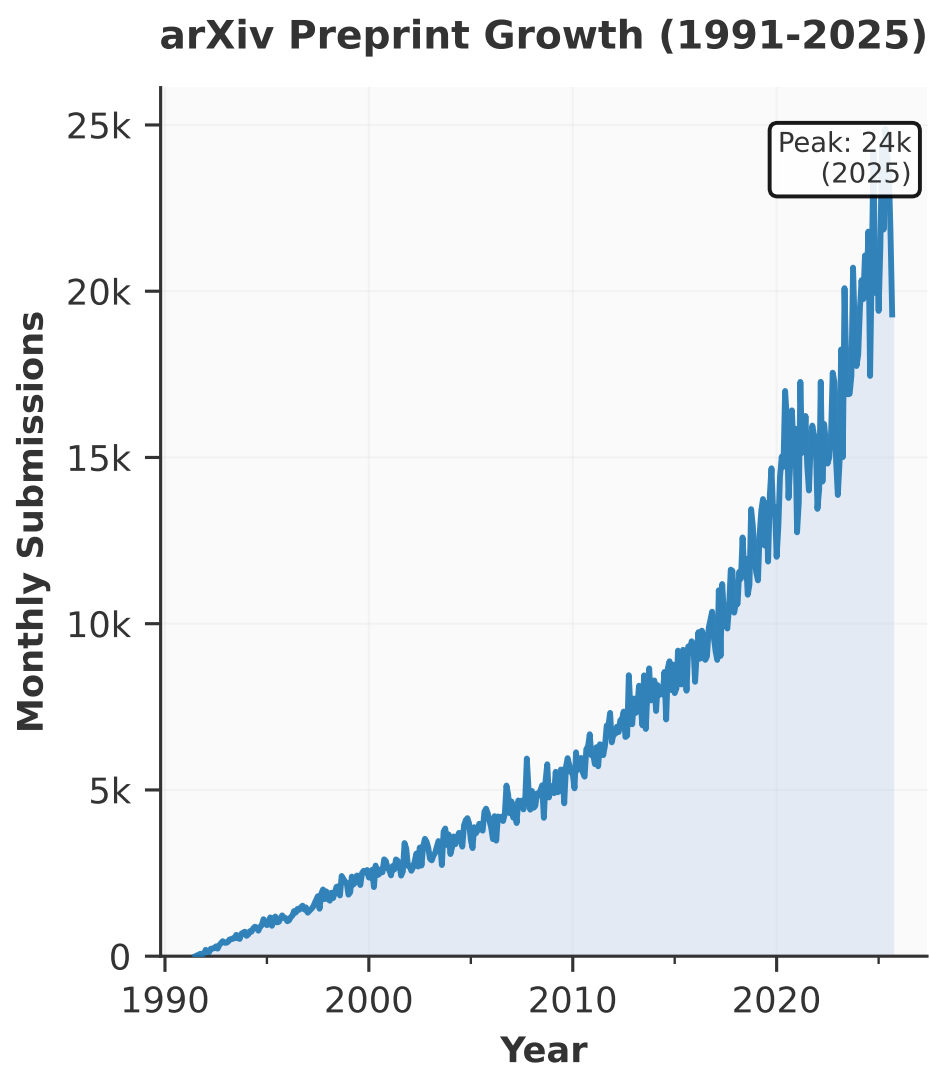
For inline mathematics, you can use single dollar sign delimiters ( $\$ . . \$$ ), allowing formulas like  $E = mc^2$  to be embedded directly within your text. For larger, display-style equations, you can use double dollar signs ( $\$\$ . . \$\$$ ) to centre the expression on its own line. For example:

$$i\hbar \frac{\partial}{\partial t} \Psi(\mathbf{r}, t) = \hat{H} \Psi(\mathbf{r}, t)$$

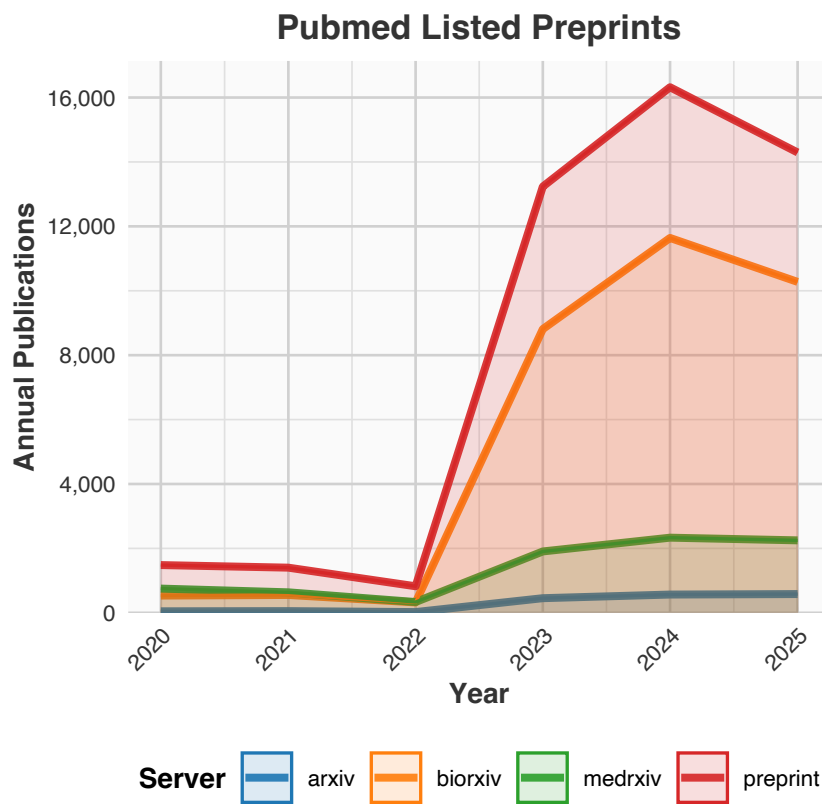
The framework’s multi-pass translator is designed to protect these mathematical expressions, ensuring they are not accidentally altered during the conversion from Markdown to LaTeX. This robust system supports a wide range of mathematical and statistical notation, from simple symbols to complex multi-line equations, ensuring your quantitative work is always presented clearly and professionally.

**Supp. Note 5: Intelligent Caching and Validation.** To accelerate compilation, Rxiv-Maker uses an intelligent caching system that avoids redundant work. It operates by generating a checksum—a unique digital signature—for each figure’s dependencies, including the source script and any input data files. A figure is only regenerated if this checksum changes, meaning you can recompile your manuscript rapidly during writing, as only the modified components are rebuilt. This saves considerable time without sacrificing the reproducibility of the final document. Complementing this is a powerful validation framework that acts as a quality-control mechanism. Running `rxiv validate` performs a multi-level check of your manuscript. Before compilation, it looks for missing figure files, broken cross-references, and malformed bibliography entries. During compilation, it parses LaTeX logs to provide clear, understandable error messages. After compilation, it even performs a lightweight scan of the PDF to flag potential rendering issues. This ensures your manuscript is technically sound at every stage.

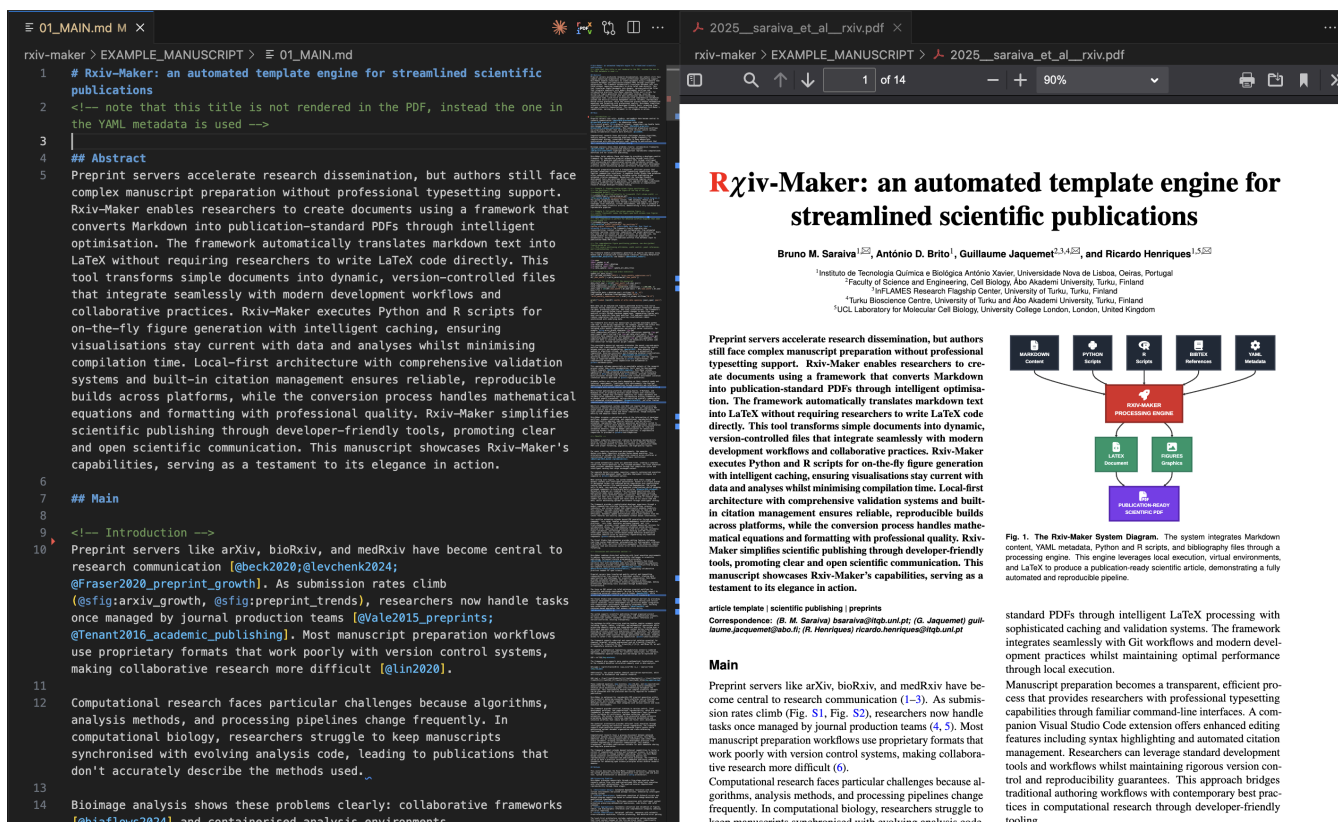




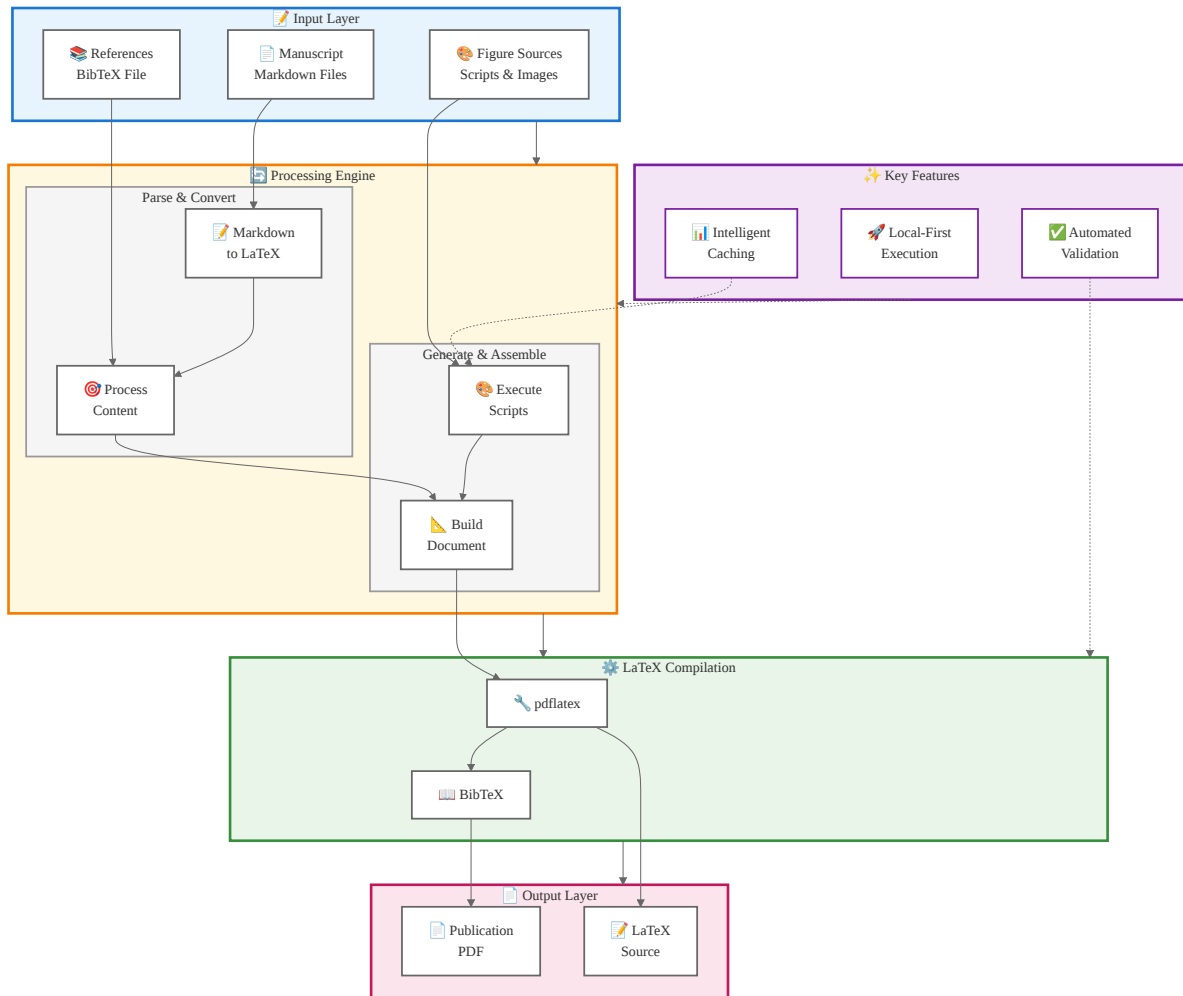
**Sup. Fig. S1. The growth of preprint submissions on the arXiv server (1991-2025).** This figure was generated from public arXiv statistics using a Python script executed by the Rxiv-Maker pipeline, demonstrating reproducible, data-driven visualisation.



**Sup. Fig. S2. Preprint Submission Trends Across Multiple Servers (2018-2025).** This figure, showing preprints indexed by PubMed from major repositories, was generated from public data (27) using a reproducible R script within the Rxiv-Maker pipeline.



**Sup. Fig. S3. The Rxiv-Maker Visual Studio Code Extension.** The extension enhances the authoring experience by providing syntax highlighting for extended Markdown, autocompletion for citation keys (e.g., @Knuth...), and real-time validation to catch errors as you type.



**Sup. Fig. S4. Detailed System Architecture.** A comprehensive technical diagram of the Rxiv-Maker architecture, illustrating the input layers, the processing engine's components (parsers, converters, generators), the compilation infrastructure, and the final output generation. This figure highlights the modular design that enables robust and reproducible manuscript compilation.