The Journal of Open Source Software

stan-playground: Run Stan models directly in your browser

Brian Ward [1*¶], **Jeff Soules** [1*], **and Jeremy Magland** [1*]

1 Center for Computational Mathematics, Flatiron Institute ¶ Corresponding author * These authors contributed equally.

DOI: 10.21105/joss.09531

Software

- Review
- Repository
- Archive

Editor: Neea Rusch

Reviewers:

- @g-rppl
- @jflournoy

Submitted: 28 October 2025
Published: 10 February 2026

Summary

Stan Playground is an in-browser editor and execution environment for Stan (Carpenter et al., 2017; Stan Development Team, 2025) statistical models. Models and analyses run entirely locally with no installation required. The only non-local computation required is on-demand compilation, which is performed by a bespoke server that translates users' models into WebAssembly modules built with the Emscripten (Zakai, 2011) toolchain. The server is available as code or a Docker image, and a free, public server is provided, making this process transparent to the end user while still allowing for customization. Built-in sharing features and integration with Pyodide (The Pyodide development team, 2021) and webR (Stagg et al., 2023) make Stan Playground a complete environment for instruction, prototyping, and analysis.

Statement of need

Stan is a language for statistical modeling commonly used for Bayesian data analysis in many fields. Running a Stan program consists of two steps: model *compilation*, where the users' Stan code is transpiled to C++ and compiled by a C++ toolchain, and *sampling*, where the resulting model is executed to perform statistical inference.

Because the models are themselves user-provided, model compilation requires end users to have a working C++17 development environment installed in order to use Stan. This requirement can be difficult to satisfy for users from less technical backgrounds or with IT-imposed constraints on their systems. As a result, installation-related questions are ubiquitous on the Stan forums. Installation issues can be a particular pain point in learning environments, where instructors do not want to spend a substantial fraction of their teaching time just to ensure students have a working environment and copy of the relevant data and models.

Thanks to its browser-based design requiring no installation, Stan Playground allows users to immediately focus on the issues of modeling and analysis that matter to them. A user simply navigates to https://stan-playground.flatironinstitute.org/ to get a blank project, or a student clicks on a link provided by their instructor with code and data already populated. They can then begin editing, compiling, and running Stan models immediately, regardless of their local system configuration.

A similar level of convenience could be achieved by performing both compilation and sampling on a centrally provided server, but there are several issues with this approach. Sampling involves executing user-written code that could perform potentially-unbounded amounts of compute. Offering this as a public service would require authentication or rate-limiting. Additionally, even in a restricted language like Stan, one would have to take serious precautions before allowing untrusted user code to run unchecked.

By contrast, providing a public server that only performs compilation is secure and practical. Compilation times are similar for most models and compilation results can be cached, meaning e.g. students in a classroom setting would only end up compiling the provided example once collectively. Furthermore, compilation never executes any user-controlled code, obviating most security concerns.

With Stan Playground, getting started with Stan can be reduced from a lengthy, sometimes quite technical, process into loading a web page in any modern browser.

Key features

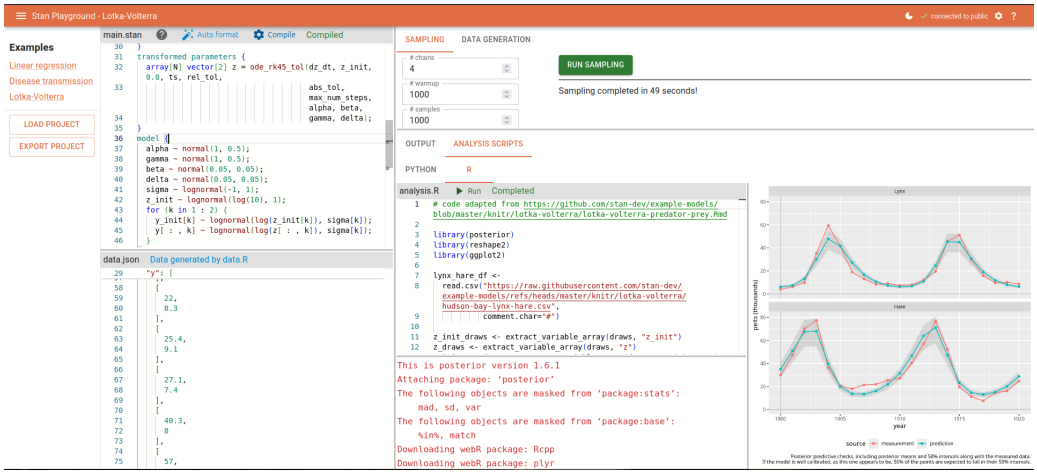

Figure 1: Stan Playground's UI after running the Lotka-Volterra example

Stan Playground provides many features to allow end-to-end analysis projects to be completed in the browser.

1. **Data Preparation**. The input data for the Stan model can be provided manually as JSON or generated by Python or R scripts which can download existing datasets or dynamically generate synthetic data.
2. **Model Editing**. The Stan editor in Stan Playground features integration with the Stan compiler to provide on-the-fly diagnostics and formatting.
3. **Compilation**. A premier feature of Stan Playground is the ability to compile and run without needing additional tools installed. This is achieved through a server that handles this compilation and provides WebAssembly modules back to the client. While a public server is hosted by the Flatiron Institute, users can also run their own and connect to that instead in the settings UI.
4. **Sampling**. Users can control the key parameters for the sampling algorithm (number of iterations, etc) and monitor progress in real time.
5. **Posterior Analysis**. Stan Playground provides many built-in diagnostics and visualizations for the posterior sample. Additionally, users can write Python or R scripts to leverage the existing ecosystem of tools such as ArviZ (Kumar et al., 2019) or `posterior` (Bürkner et al., 2025).
6. **Sharing and Export**. Stan Playground by default saves the project in browser local storage. For longer-term storage, users can export their projects to zip files for local use, or upload to a GitHub Gist. GitHub Gist links facilitate easy sharing, as they can be used

in URL parameters to load a specific project when a link is clicked. More customizable sharing options are also available through URL parameters.

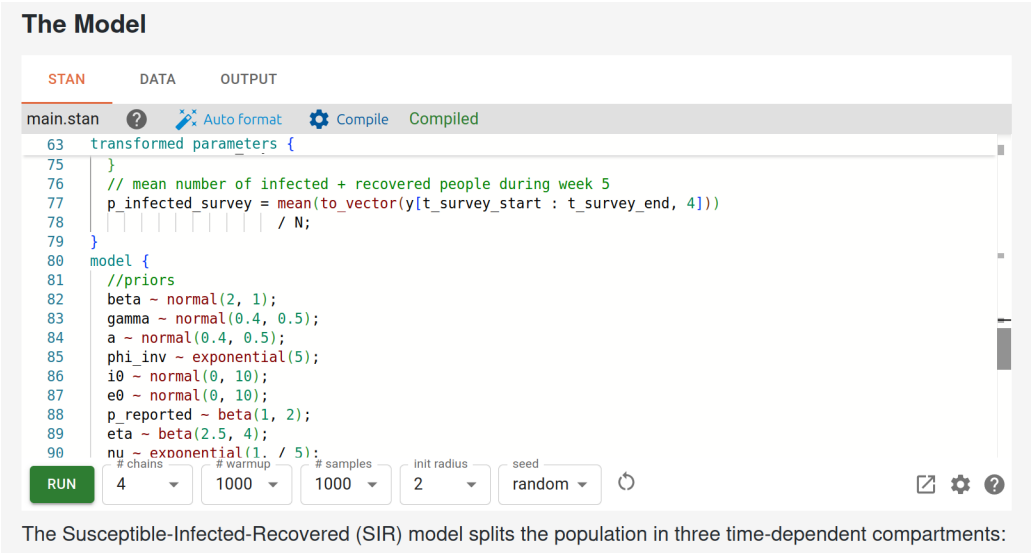

Figure 2: Stan Playground's embedded layout inside a mock document. Model from Grinsztajn et al. (2021).

7. **Embedding**. Stan Playground is able to be fully embedded using the `<iframe>` tag. A second UI layout which is more amenable to embedding is provided, and a custom element to make this easier is available in a small, no-dependency script. Blogs or documentation sites can use these to gain interactivity with minimal effort.
8. **Examples**. Stan Playground provides several example programs on the left sidebar of the home page at various levels of complexity, spanning from a basic linear regression on fake data to a reimplementation of the model and analysis from Carpenter (2018). The linked example uses Stan Playground's sharing features to pre-populate the project on page load.

Implementation details

Stan Playground consists of two components. The first is a server that can compile Stan models to WebAssembly, implemented in Python and leveraging the existing Emscripten toolchain (Zakai, 2011). This server is also published as a Docker image for users who wish to run a local instance.

The remainder of the project is the user-facing web application, which can be found at https://stan-playground.flatironinstitute.org/.

This web application is built on the React framework and features various panels to allow editing of Stan models, compilation, sampling, and downstream analysis of the results. The use of WebAssembly and browser Web Workers to move computations off the main thread allows these computations to complete quickly and without freezing the user interface. Users or developers can run the web frontend locally by installing Yarn and following the instructions provided in the repository.

Similar projects

Several projects have previously employed the Web platform for assisting Stan users, including ShinyStan (Gabry & Veen, 2025) and MCMC Monitor. However, both of these tools relied on an existing, local installation of Stan to perform sampling, inheriting the usual installation difficulties as a result.

The idea of a web-based "playground" for a programming language is also well-explored territory. Some examples that inspired the authors in various ways include:

- The Rust Playground (play.rust-lang.org).
- The Compiler Explorer (godbolt.org).
- JSFiddle (jsfiddle.net).

Note that while Stan Playground shares many conceptual ideas with these previous tools, the technical details differ greatly; for instance, both Compiler Explorer and Rust Playground provide code execution via hosted server environments rather than the use of WebAssembly.

Furthermore, the integration of a full probabalistic programming environment, with data preparation, plotting, and more on top of sampling the model, remains a unique contribution by this project.

Acknowledgements

Andrew Gelman and Jonah Gabry from Columbia University offered valuable early feedback on this project.

This project relies on `tinystan` (Ward, 2025), which was itself inspired by Edward Roualdes' `BridgeStan` project (Roualdes et al., 2023).

References

Bürkner, P.-C., Gabry, J., Kay, M., & Vehtari, A. (2025). *posterior: Tools for working with posterior distributions*. https://doi.org/10.32614/cran.package.posterior

Carpenter, B. (2018). Predator-Prey Population Dynamics: the Lotka-Volterra model in Stan. *Stan Case Studies*, *5*. https://mc-stan.org/learn-stan/case-studies/lotka-volterra-predator-prey.html

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., & Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of Statistical Software*, *76*(1). https://doi.org/10.18637/jss.v076.i01

Gabry, J., & Veen, D. (2025). *ShinyStan: Interactive visual and numerical diagnostics and posterior analysis for Bayesian models*. https://doi.org/10.32614/cran.package.shinystan

Grinsztajn, L., Semenova, E., Margossian, C. C., & Riou, J. (2021). Bayesian workflow for disease transmission modeling in Stan. *Statistics in Medicine*, *40*(27), 6209–6234. https://doi.org/10.1002/sim.9164

Kumar, R., Carroll, C., Hartikainen, A., & Osvaldo, M. (2019). ArviZ a unified library for exploratory analysis of Bayesian models in Python. *Journal of Open Source Software*. https://doi.org/10.21105/joss.01143

Roualdes, E. A., Ward, B., Carpenter, B., Seyboldt, A., & Axen, S. D. (2023). BridgeStan: Efficient in-memory access to the methods of a Stan model. *Journal of Open Source Software*, *8*(87), 5236. https://doi.org/10.21105/joss.05236

Stagg, G. W., Lionel, H., & Others. (2023). *webR: The statistical language R compiled to WebAssembly via Emscripten* (Version 0.2.2). https://github.com/r-wasm/webr

Stan Development Team. (2025). *About Stan*. https://mc-stan.org/

The Pyodide development team. (2021). *Pyodide/pyodide* (Version 0.28.3). Zenodo. https://doi.org/10.5281/zenodo.17177884

Ward, B. (2025). tinystan: Easy, minimal interface to the stan samplers in several languages. In *GitHub repository*. GitHub. https://github.com/WardBrian/tinystan

Zakai, A. (2011). Emscripten: An LLVM-to-JavaScript compiler. *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, 301–312. https://doi.org/10.1145/2048147.2048224