PharmaSUG 2026 - Paper ET-378

AI-Enhanced R Shiny App for Real-Time Clinical TLF Coding and Preview

Dickson M Wanjau

Merck & Co., Inc., Rahway, NJ, USA

ABSTRACT

In this paper, we present an interactive R Shiny application that streamlines the creation, preview and refinement of RTF-based TLFs for clinical reporting using the open-source “r2rtf” R package. The application merges a live code editor with a real-time PDF preview, enabling users to iteratively write or modify code and immediately preview results. Users begin by uploading “table-ready” datasets in various formats e.g. CSV, Excel, SAS (.xpt or .sas7bdat formats), R file formats (.RData, .rda) etc. The app previews data, offers column filtering, and auto-generates starter r2rtf code which users can refine in a shinyAce editor. Upon code submission, the app executes the user’s script in a controlled environment, updates the preview with a rendered PDF and logs any warnings or errors for debugging.

To support adoption and learning, we have embedded a retrieval-augmented generation (RAG) AI chatbot trained specifically on the “r2rtf” package documentation. This facilitates on-demand guidance about functions, syntax and best practices while building submission-ready tables. Additionally, the user can debug some common syntax errors or warnings. Together, these features lower the barrier to TLF creation in R, enhance transparency and reproducibility and provide a unified interface for programmers across experience levels. We discuss implementation logic, user workflow, error handling strategies and integration of AI-assistance and we demonstrate real clinical examples to illustrate practical value. The tool supports reproducible reporting and enhances productivity for statistical programmers.

INTRODUCTION

Production of Tables, Listings, and Figures (TLFs) is a core activity in clinical trial reporting. While R adoption continues to grow, many clinical programmers still face challenges with:

- table formatting complexity driven by complex TLFs specifications,
- slow “edit → render → inspect” cycles that impede rapid iteration especially for large listings which can number to upwards of 100 listings required
- onboarding and training barriers for newer open-source packages and tools (IDEs)

The r2rtf R package provides a structured workflow for assembling RTF tables and figures through simple “verb” functions corresponding to logical table components such as titles, column headers, body, footnotes, and source information. Designed to work with base R pipes (|>) and “magrittr” R package pipes (%>%) semantics, these functions allow programmers to translate data frames into publication-ready RTF outputs while retaining flexible control over formatting and structure.

Some commonly used “verb-layer” functions include:

- rtf_title(): RTF title information
- rtf_colheader(): RTF column header information
- rtf_body(): RTF table body information
- rtf_footnote(): RTF footnote information
- rtf_source(): RTF data source information
- rtf_page(): RTF page information

The “r2rtf” package offers extensive flexibility for defining and formatting production-ready Rich Text Format (RTF) tables and figures.

It supports highly customizable table layouts through a simple set of user-friendly parameters and clearly structured data requirements.

Users can define complex column headers with a **delimiter ("|")** and must specify only three core inputs for basic outputs**: the dataset, output filename** and **column relative widths**.

The package also provides detailed control over structural and aesthetic table properties at the cell, row, column, and overall table levels.

Formatting options include: a variety of **border styles (e.g., single, double, dashed)**, **text alignment (left, right, center, decimal)**, **adjustable column widths**, and **text appearance properties such as bold, italics, underline**, and combinations thereof.

Users can also control **font size**, **text** and **border colors** (with a large palette accessible via the color() function), and support for **special UTF-8 characters** (e.g., Greek, symbolic, East Asian scripts).

Additionally, r2rtf enables appending multiple tables into a single file and supports pagination.

However, building complex tables, listings and figures still often require iterative trial-and-error. This paper introduces an interactive Shiny application that accelerates r2rtf-based table development through:

- real-time code execution and preview
- guided table header editing tools,
- and embedded RAG-based AI assistance

Proprietary

OVERVIEW OF THE SOLUTION

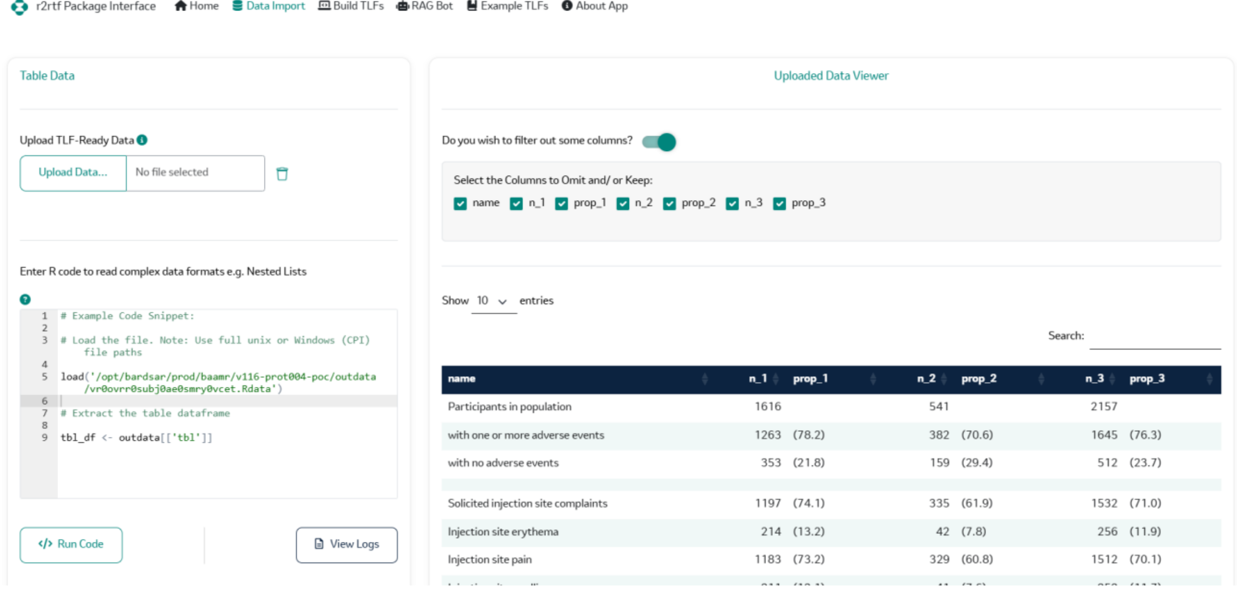
The proposed application provides a streamlined, end-to-end workflow for generating production-ready Tables, Listings, and Figures (TLFs) using an interactive interface. The workflow is designed to guide users from raw input data to finalized output with minimal friction and iterative feedback at each step. A high-level overview of the process is described below, with corresponding screenshots provided to illustrate the user interface.

TLF Data Upload

Users can upload tlf-ready datasets or execute R / Python/ SAS code (if remote execution via a SAS server is enabled) code to ingest data directly within the application. Supported formats include CSV, TXT, Excel formats (.xls, .xlsx), SAS (SAS7BDAT and XPT), and R data formats (RDS and RData), enabling flexibility across common clinical data sources.

- **Data Preview and Column Selection (Optional)**

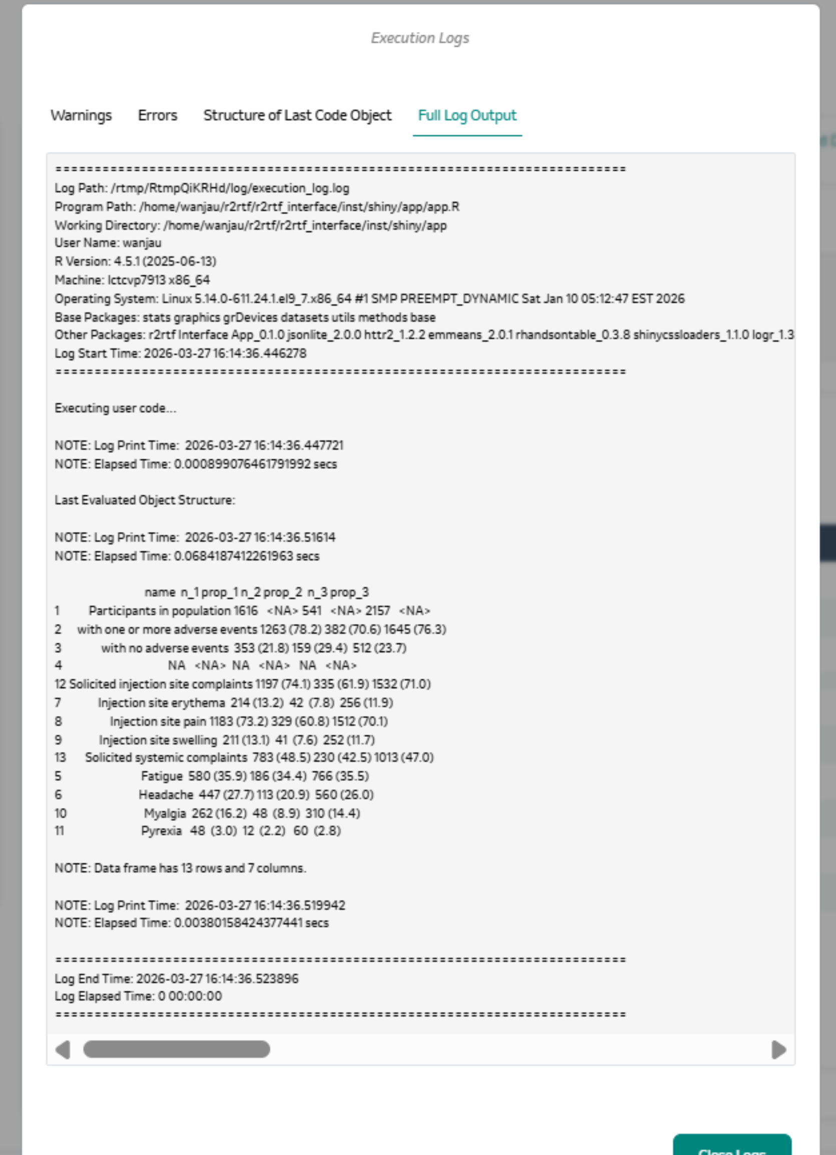
The application provides an interactive data preview along with a checkbox-based interface that allows users to selectively include or exclude columns. This enables quick refinement of the dataset prior to table generation.



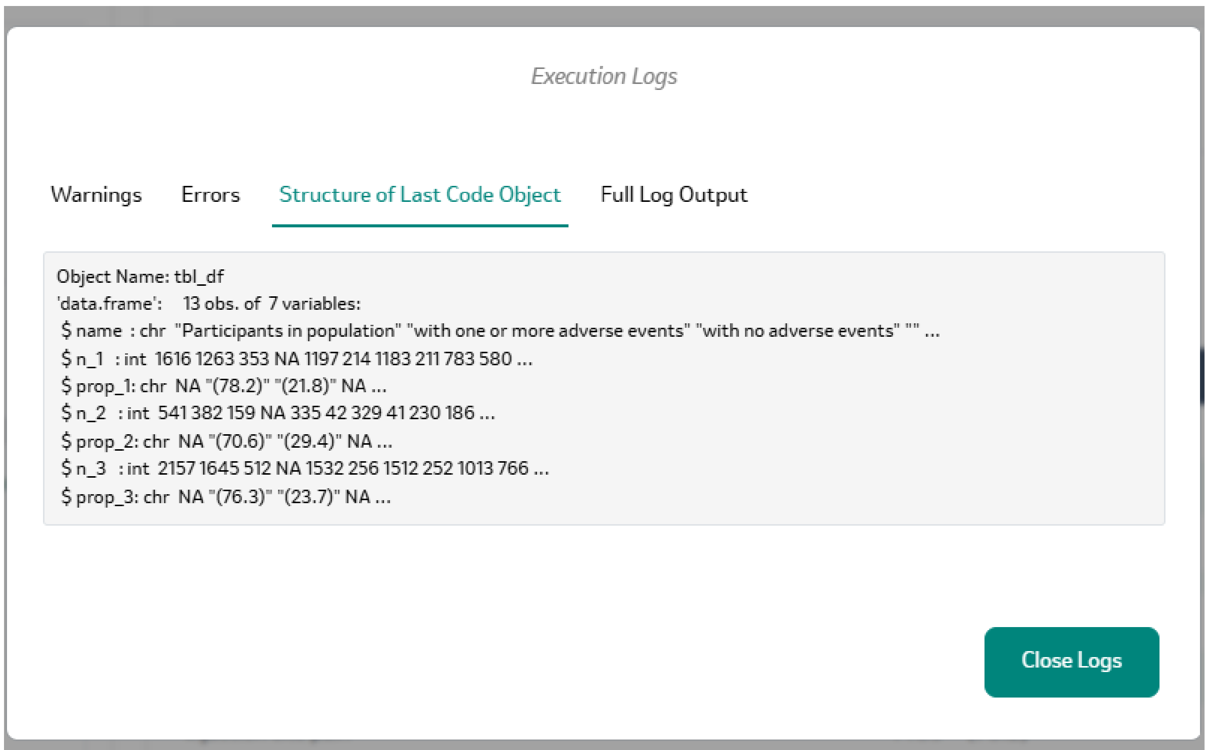
Logging Capability

Additionally, the application incorporates a structured logging capability leveraging the logrx package. This enables real-time capture and display of execution logs, including errors and warnings related to file

access, directory permissions, and data ingestion issues. For example, users are notified if a file cannot be read due to path or permission constraints, or if inconsistencies arise during data loading.



To further support data validation, users can inspect the structure of the uploaded dataset within the application, allowing them to verify variable types, formats, and overall data integrity prior to table generation. This ensures that downstream processing with r2rtf is based on correctly structured inputs.



Automatic Code Generation

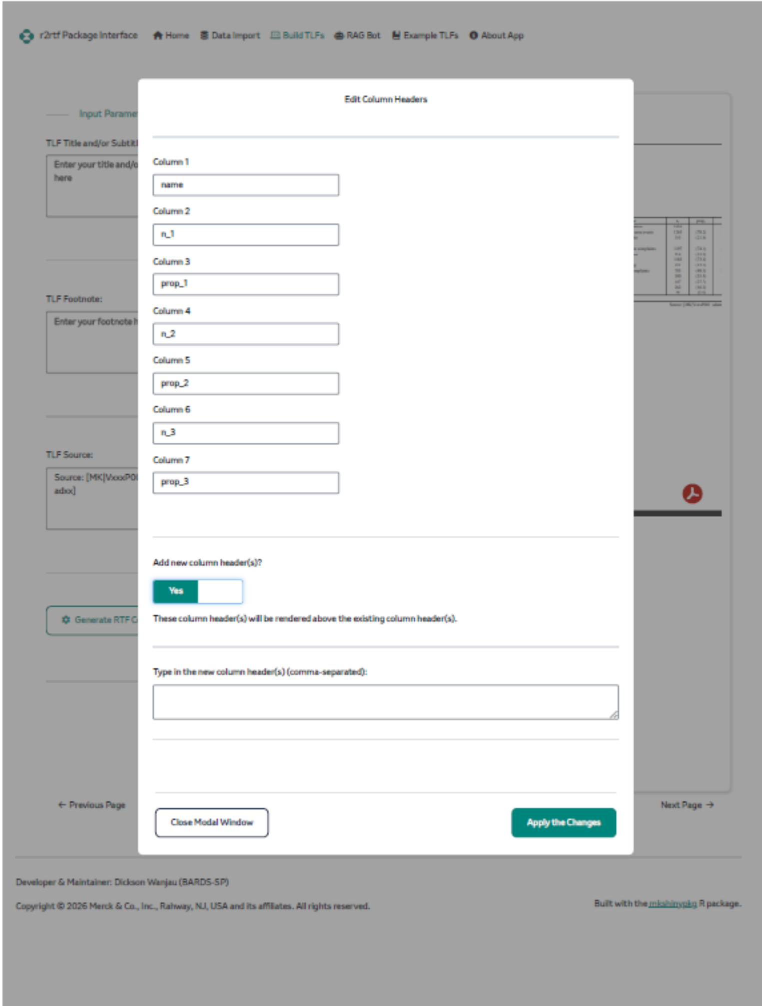
Based on the selected dataset, the application generates a runnable starter script using standard r2rtf verbs, including rtf_title(), rtf_colheader(), rtf_body(), rtf_footnote(), and rtf_source(). This code is automatically populated in the editor, providing a structured starting point for further customization.

Additionally, users are provided with input fields for commonly used elements, including the TLF title, subtitle, footnote, and source.

Interactive Code Editing

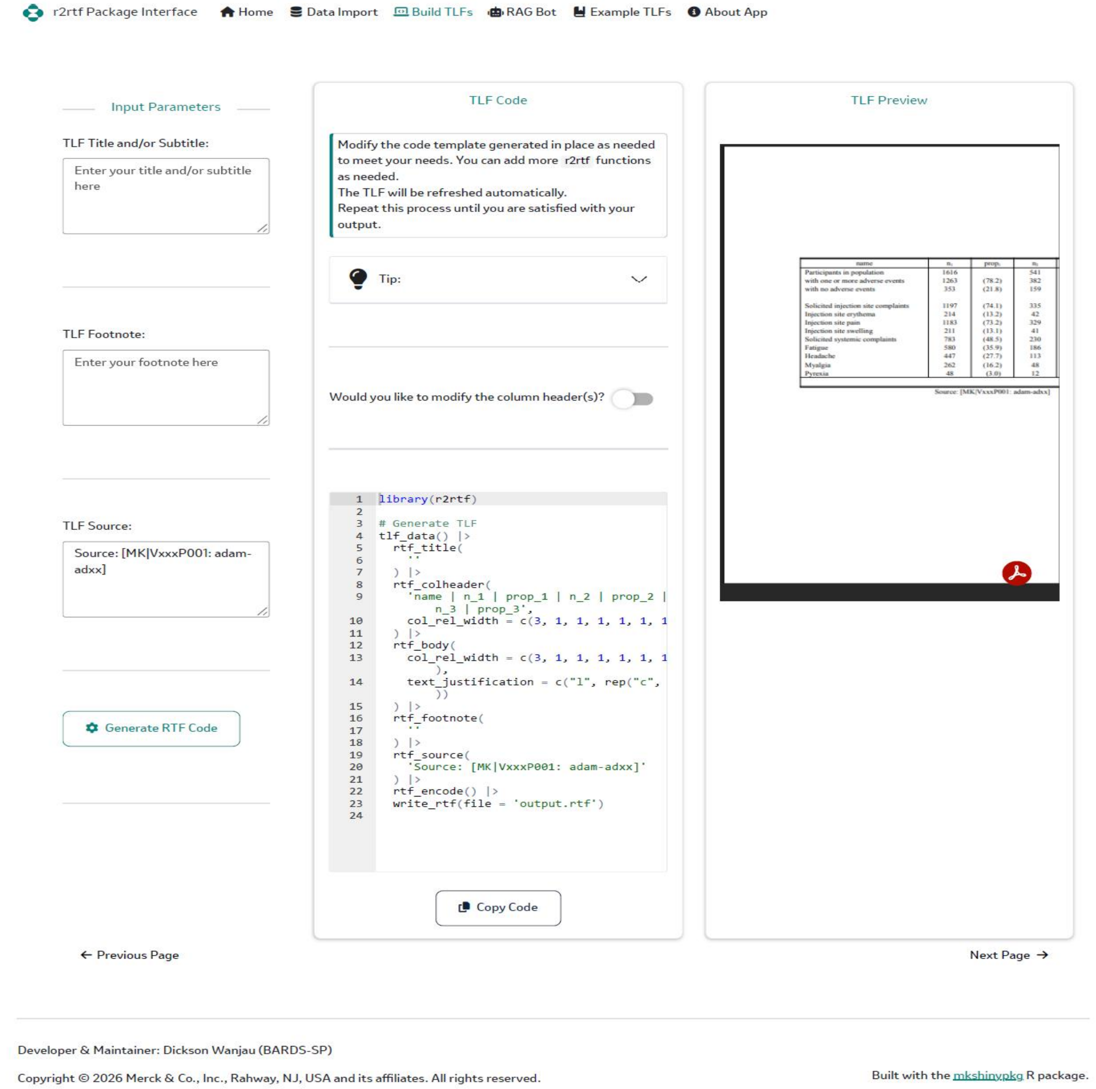
Users can modify the generated script within an embedded code editor powered by shinyAce. The editor supports syntax highlighting and reactive updates, enabling an experience similar to working in a traditional integrated development environment.

Additionally, the application provides a modal dialog interface that allows users to modify existing column headers or define new ones, with the option to specify how new headers span across columns in the TLF.



Real-Time Preview and Rendering

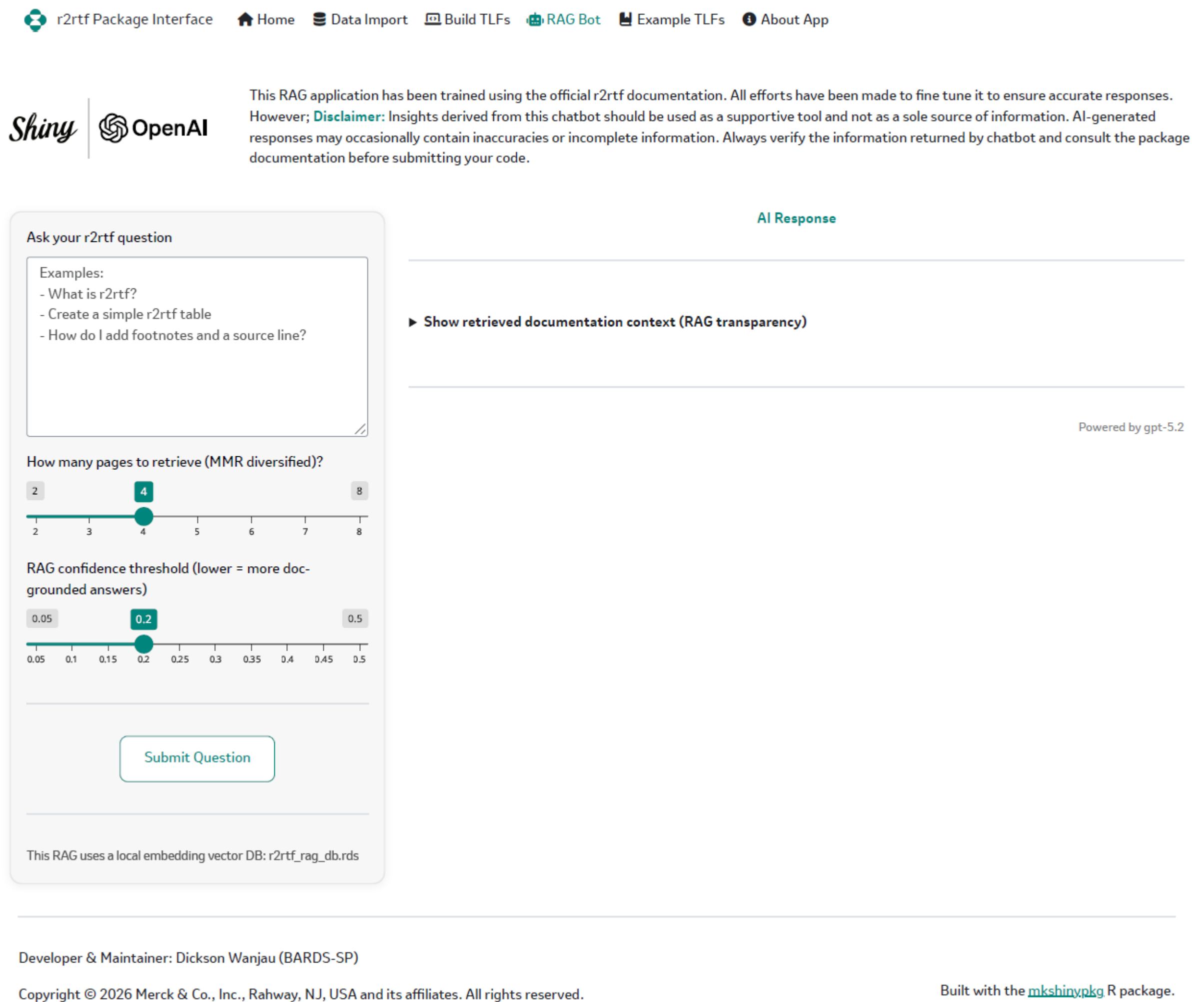
Upon execution, the user's code is evaluated in a controlled environment. The resulting output is rendered to RTF, converted to PDF, and displayed directly within the application using an embedded iframe. This enables rapid "edit → render → inspect" cycles without leaving the interface.



AI-Assisted Guidance

To further enhance usability, the application includes a retrieval-augmented generation (RAG) chatbot powered by the OpenAI GPT-5.2 model, providing contextual, real-time assistance to users. The assistant is grounded in official r2rtf documentation, including package vignettes, CRAN resources, and supporting materials such as R for Clinical Study Reports and Submission. This enables users to receive accurate guidance on syntax, functions, and best practices while developing TLFs.

Proprietary



The underlying design and implementation of the RAG framework are discussed in detail in the following sections.

METHODS / IMPLEMENTATION

Data Handling and Validation

The TLF Data is managed using reactive containers (*reactiveVal*, *reactive*, *reactiveValues*) to ensure seamless propagation of updates throughout the application. Uploaded files are automatically detected by their extension and ingested accordingly, supporting CSV, TXT, Excel, SAS (SAS7BDAT, XPT), and R formats (RDS, RData).

```
observeEvent(input$file, {
  req(input$file)
  file <- input$file$datapath
  ext <- tools::file_ext(file)

  data <- tryCatch({
    switch(ext,
           "csv"  = read.csv(file, stringsAsFactors = FALSE),
           "txt"  = read.table(file, header = TRUE, sep = "\t",
stringsAsFactors = FALSE),
           "xlsx" = readxl::read_excel(file),
           "xls"  = readxl::read_excel(file),
           "rds"  = readRDS(file),
           "rda"  = {
             data_env <- new.env()
             load(file, envir = data_env)
             get(ls(data_env)[1], envir = data_env)
           },
           "RData" = {
             data_env <- new.env()
             load(file, envir = data_env)
             get(ls(data_env)[1], envir = data_env)
           },
           "sas7bdat" = read_sas(file),
           "xpt" = read_xpt(file),
           stop("Unsupported file format!")
    )
  }, error = function(e) {
    showNotification(HTML(paste("Error:", e$message)), duration = 10, type =
"error")
    NULL
  })
```

The application validates the structure of the uploaded data, allowing users to inspect the data to confirm correct data types and column formats.

Errors, warnings, or access issues such as missing files or read permission failures, are captured using *tryCatch()* and surfaced immediately through notifications and an integrated logging system, improving transparency and reducing debugging time

Logging Strategy

A modal log viewer consolidates errors, warnings, and the structure of the last evaluated object. Full log output is provided via the logrx package, improving traceability and reproducibility while simplifying debugging. Users can quickly identify issues with file access, data integrity, or code execution without leaving the application.

Code Generation Strategy

The application generates structured and reproducible code using the currently filtered data object.

Column headers, titles, subtitles, footnotes, and source references are automatically populated to minimize repetitive coding tasks.

Users can modify existing column headers or define new upper-level spanning headers via a modal dialog, which also provides embedded examples to educate users on best practices. Default widths, alignment, and formatting logic are applied to ensure consistent layouts. This structured approach reduces trial-and-error iterations and accelerates the development of complex TLFs, particularly for users new to the r2rtf workflow.

Execution Engine and Controlled Evaluation

User scripts are evaluated within a dedicated environment to isolate execution and prevent unintended side effects.

Syntax errors, warnings, and other execution messages are displayed directly in the Ace code editor in real time, replicating the experience of working in a professional IDE such as RStudio, Posit, or VSCode.

Reactive execution ensures that updates to data, headers, or code are immediately reflected in the preview, supporting iterative and interactive development.

Rendering Pipeline (RTF → PDF)

The Generated RTF output is written to a temporary directory to maintain a clean development environment. The .rtf file is automatically converted to pdf to enable preview.

The most-recent PDF is embedded via:

```
tags$iframe(type="application/pdf", src="...#toolbar=0")
```

This provides a live preview without providing to the user to download output during development to enforce traceability.

Once satisfied with the output, users can copy the generated code to their IDE or workflow using a “Copy to Clipboard” button, enabling seamless integration into standard clinical programming pipelines.

EMBEDDED AI (RAG) ASSISTANT

Motivation

Users often need quick, contextual guidance while developing TLFs in R with the r2rtf package. Common requests include valid function arguments, column spanning rules, alignment logic, and expected object types. To address these needs, the application integrates a retrieval-augmented generation (RAG) assistant powered by the OpenAI GPT-5.2 model.

How it works

The RAG assistant is built on a locally curated knowledge base of the r2rtf package documentation. The content is collected by scraping official reference pages, package vignettes, and supporting resources such as R for Clinical Study Reports and Submission. The scraped text is segmented into manageable chunks, cleaned, and converted into embeddings using OpenAI's *text-embedding-3-small model*. These embeddings form a semantic index, enabling the assistant to quickly retrieve relevant documentation for a user's query.

When a user submits a question, the system generates an embedding of the query and computes cosine similarity scores against the indexed chunks. A Maximal Marginal Relevance (MMR) algorithm selects the most relevant and diverse chunks, reducing redundancy and ensuring the assistant considers multiple perspectives. The retrieved context is then prepended to a system prompt and sent to the GPT-5.2 model, which generates documentation-grounded responses. If the retrieved context is insufficient or the similarity falls below a threshold, the assistant can fall back to general knowledge about r2rtf while still prioritizing safe and accurate guidance.

The RAG assistant provides concise, actionable answers and can include runnable R code snippets when requested. Sources for each response are explicitly listed, and users are informed whether the answer relied on retrieved context or general knowledge. This design minimizes hallucinations, accelerates learning, and empowers users to debug or refine TLF code in real time while maintaining adherence to best practices.

Technical Implementation

Maximal Marginal Relevance (MMR) for Context Retrieval

When a user asks the RAG bot a question, the system doesn't just pick the single most similar document chunk. Instead, it uses **Maximal Marginal Relevance (MMR)** to select a small set of text snippets that are both **relevant** to the query and **diverse** relative to each other.

This helps the AI give better answers and reduces repeated or redundant content.

Here's how it works in practice:

1. **Compute similarity**: Each documentation chunk in the RAG database is embedded as a numeric vector. The query from the user is also embedded, and a cosine similarity score is calculated between the query and each chunk. This measures relevance.
2. **Iterative selection**: The first chunk selected is the one most similar to the query. Subsequent chunks are chosen by balancing:
   - **Relevance**: How similar the chunk is to the user's question
   - **Redundancy penalty**: How similar the chunk is to already selected chunks

Mathematically, each candidate chunk's **MMR score** is:

$$\text{MMR} = \lambda * \text{Relevance} - (1 - \lambda) * \text{MaxRedundancy}$$

where λ $(alpha)$ controls the tradeoff between relevance and diversity.

3. **Context assembly**: The top chunks selected by MMR form the **documentation context** that is fed to the AI model, ensuring answers are both accurate and informative, without repeating the same text.

By using MMR, the RAG assistant can provide **documentation-grounded guidance** that is concise, relevant, and covers multiple facets of a user's question.

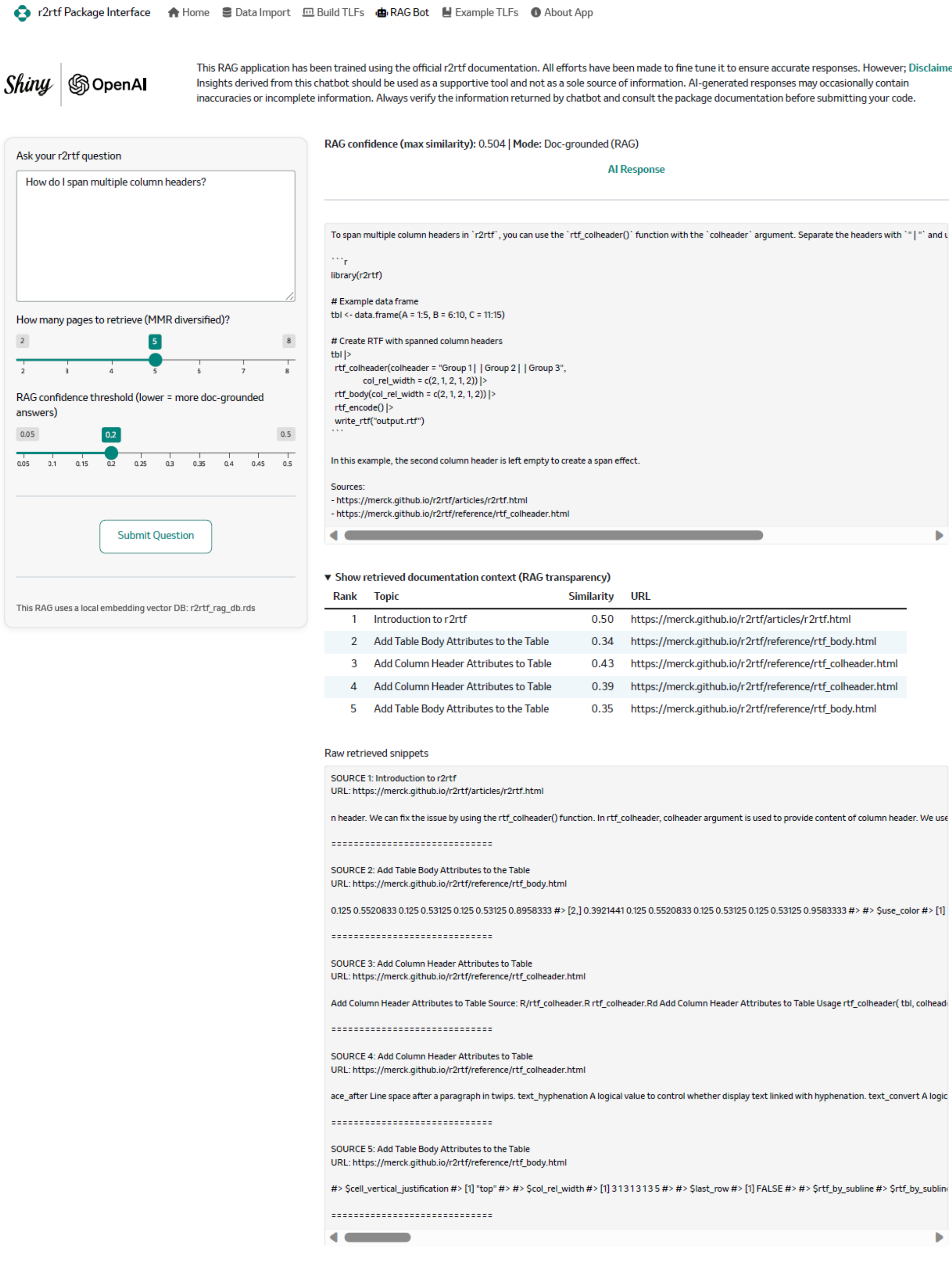
The application allows users to fine-tune the retrieval behavior using two sliders:

1. **Pages to retrieve (MMR diversified) –** This controls how many document chunks are retrieved for context. A higher number increases coverage and diversity, while a lower number keeps responses more focused
2. **RAG confidence threshold –** This sets a similarity threshold for using documentation in the AI response. Lower thresholds force the model to rely more strictly on retrieved documentation, producing **documentation-grounded answers**; higher thresholds allow the model more freedom to generate text based on general knowledge

Together, these controls let users adjust the balance between **relevance, diversity and creativity**, enabling tailored guidance grounded in official r2rtf documentation and supporting resources.

Proprietary

Sample Query and corresponding RAG response:



DISCUSSION

The application provides several key benefits for clinical programming workflows. Productivity is enhanced as users require fewer iterations to reach final table formatting, thanks to real-time preview and auto-generated starter code. The barrier to entry for new users is lowered, as the embedded RAG assistant provides contextual guidance on r2rtf syntax, column spanning, alignment, and expected data types. Reproducibility is supported by allowing users to copy or export the generated code directly into their existing workflows, while transparency is maintained through detailed logging and inspection of object structures, which supports auditing and troubleshooting.

This application exemplifies emerging trends in clinical programming by combining open-source reporting tools, interactive application frameworks, and AI-driven documentation guidance. It demonstrates the interoperability of r2rtf for RTF table generation, Shiny for reactive UI and live previews, and retrieval-augmented generation (RAG) AI for knowledge-grounded support, providing a unified platform that accelerates TLF development while enhancing reliability and user confidence.

LIMITATIONS AND FUTURE ENHANCEMENTS

While the current implementation streamlines RTF-based reporting, several enhancements are planned. Broader output support, including DOCX and HTML previews, would extend the utility of the application beyond RTF tables. The AI assistant can be further improved with additional guardrails, such as a citations-only response mode, stricter retrieval thresholds, and advanced RAG fine-tuning techniques to further reduce potential hallucinations and improve documentation grounding. Additional workflow automation, support for multi-user collaboration, and integration with version control systems are also under consideration for future development.

CONCLUSION

This paper introduces an AI-enhanced Shiny interface that modernizes the development of r2rtf-based TLFs by combining real-time code execution, live preview, and embedded documentation guidance. By reducing friction in iterative formatting, accelerating learning for new users, and supporting reproducibility and transparency, the application demonstrates a practical approach to leveraging open-source tools and AI in clinical reporting workflows. The solution facilitates the adoption of reproducible, efficient, and auditable processes for clinical trial table, listings and figures generation.

REFERENCES

Wang, S., Ye, S., Anderson, K. M., & Zhang, Y. (2026). *r2rtf: Easily create production-ready rich text format (RTF) tables and figures* (R package version 1.3.0) [Computer software]. https://merck.github.io/r2rtf/

Zhang, Y., Xiao, N., Anderson, K., & Zhu, Y. (2025). *R for clinical study reports and submission* [Online book]. https://r4csr.org/

Nijs, V., Fang, F., Trestle Technology, LLC, & Allen, J. (2025). *shinyAce: Ace editor bindings for Shiny* (R package version 0.4.4) [Computer software]. https://CRAN.R-project.org/package=shinyAce

Kosiba, N., Bermudez, T., Straub, B., Rimler, M., Masel, N., & Parmar, S. (2025). *logrx: A logging utility focus on clinical trial programming workflows* (R package version 0.4.0) [Computer software]. https://pharmaverse.github.io/logrx/

CONTACT INFORMATION

Name: Dickson Wanjau

Affiliation: Merck & Co., Inc. Rahway, NJ, USA

Phone: 610-202-0039

Email: dickson.wanjau@merck.com