Paper PP14

meRlin: A Code Assistant for Clinical Trial Tables

Torben Brehme, Boehringer Ingelheim Pharma GmbH & Co. KG, Biberach an der Riß, Germany
Steven Brooks, Boehringer Ingelheim (China) Investment Co., Ltd., Shanghai, China
Pietro Mascheroni, Boehringer Ingelheim Pharma GmbH & Co. KG, Biberach an der Riß, Germany
Rajan Sundar Sri Kannan, Technical University of Munich, Garching, Germany
Treesa Vadakkumpadath, University of Kassel, Kassel, Germany

ABSTRACT

Tables, listings, and figures (TLFs) are key components in clinical trial reporting, yet their generation remains a time-consuming task. Programmers often face tight deadlines and must ensure outputs are both accurate and traceable. In this work, we present an agentic workflow for automating table creation using large language models (LLMs). Our code assistant, `meRlin`, accepts requests for a clinical reporting table from the user, retrieves relevant R code templates, and accesses clinical data and trial documents such as the ADS plan and Clinical Trial Protocol (CTP). Based on these inputs, `meRlin` adapts the code templates to generate tailored R code. Programmers can then review, execute, and debug the code with the assistant's support. We illustrate the approach with reference table examples and discuss future enhancements.

INTRODUCTION

Tables, listings, and figures (TLFs) are essential components in clinical trial reporting, providing the necessary structure for communicating results and supporting regulatory submissions. However, the process of generating these outputs is often highly time-consuming, requiring programmers to navigate complex datasets and documentation while working under tight timelines. Ensuring that each output is both accurate and traceable adds another layer of complexity, making efficiency and precision critical but challenging to achieve.

In recent years, the emergence of artificial intelligence (AI) tools has offered new opportunities to streamline the drafting and generation of clinical trial documents [1], including TLFs [2]. Large language models (LLMs) and code assistants are increasingly being used to automate repetitive programming tasks, reduce manual effort, and minimize human error [3]. These AI-driven solutions can interpret user requests, retrieve relevant templates, and adapt code to specific data and documentation requirements, thereby accelerating the overall workflow and enhancing productivity for clinical programmers.

In this work, we introduce `meRlin`, an agentic code assistant designed specifically for automating the creation of TLF elements in clinical trials. `meRlin` operates by accepting user requests for specific TLF outputs, retrieving appropriate R code templates, and accessing essential clinical data and trial documents such as the ADS plan and Clinical Trial Protocol (CTP). By adapting these templates to the context of each request, `meRlin` generates tailored R code that programmers can review, execute, and debug with guided support. In the current implementation, we focus on the generation of clinical reporting tables, but in the future we plan to extend the architecture to handle other relevant TLF element types.

LLMs are few-shot learners [4], and the key idea underlying `meRlin` development is to provide the language model that is powering the agent "good" examples of how the final output should look. This requires refocusing efforts from prompt engineering to the wider engineering of the LLM context: the right piece of information (e.g., code template) should be provided to the LLM together with the piece of knowledge (e.g., the ADS plan) that is needed to generate the code for the specified TLF item.

The remainder of this paper details the methods underpinning `meRlin`'s workflow, presents reference table examples to illustrate its functionality, and discusses the results and potential future enhancements of this AI-driven approach.

METHODS

`meRlin` is built as an R `shiny` [5] application that can be run within an RStudio instance. It integrates multiple data sources and tools to support table generation workflows, as displayed in Figure 1.

`meRlin` is distributed as a company internal R package that can be installed from users through the standard `install.packages` function. Originally, `meRlin` started as a fork from the `gptstudio` [6] project, adapted to integrate access to third-party LLMs via a company approved platform. With time we introduced several changes to the original package, and now it is developed as a fully standalone project.

Once started, `meRlin` launches a non-blocking server, implemented through `fiery`, that sets up a connection to the user's R console. This server is termed *context API* in Figure 1, provides relevant local context to the LLM agent. This local context consists of metadata of dataframes in the R environment, R (or other language) code scripts, directory names and their structure. It also supports semantic search via an in-memory vector store, which is populated with

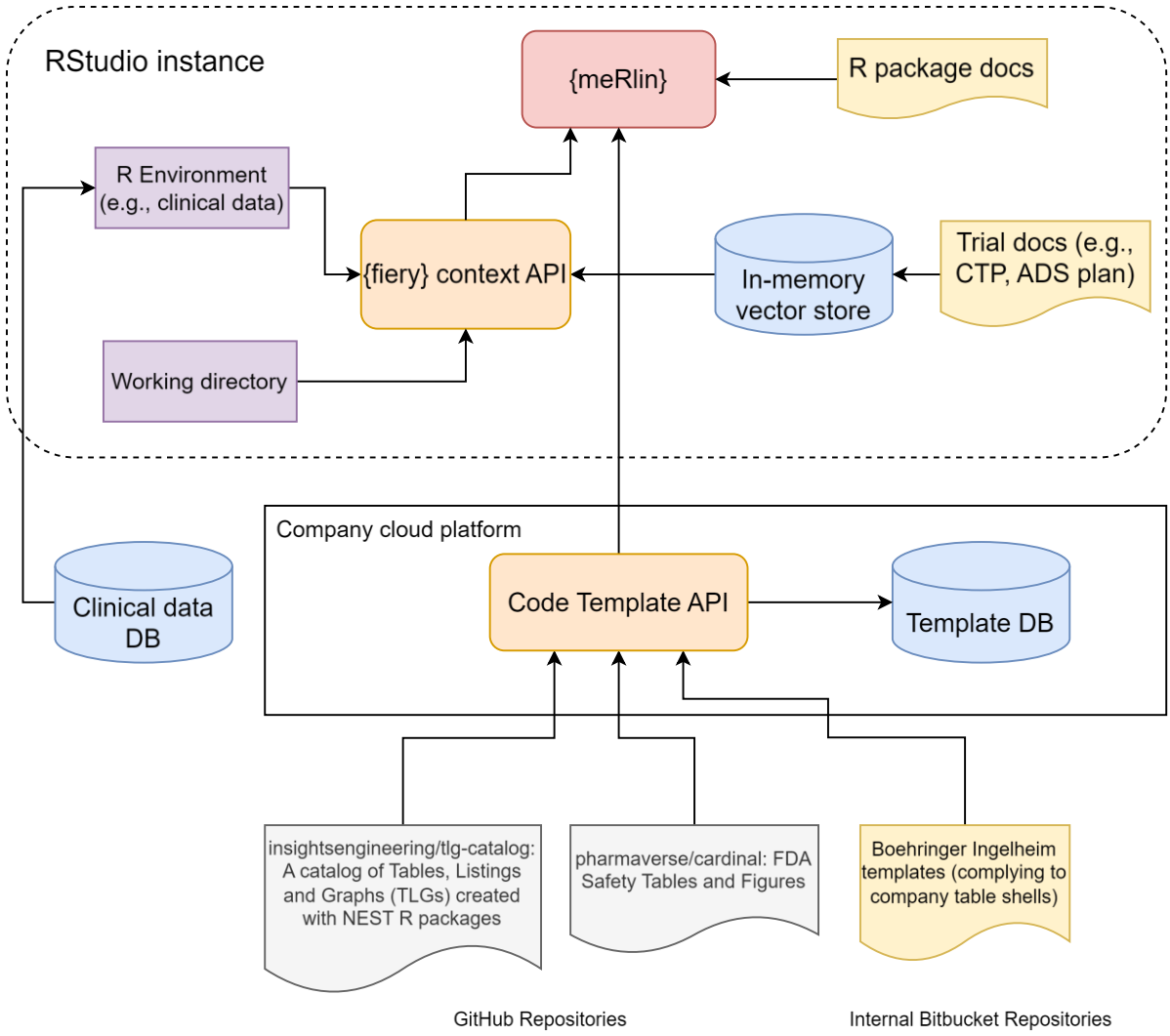

Figure 1: Schematic for the `meRlin` architecture. The agent is developed across local and remote environments. In an RStudio instance, the Context API provides the LLM powering `meRlin` with context from the R environment, the working directory and relevant trial documents. Hosted on a company cloud platform, the Code Template API fetches TLF templates from open source and company specific repositories and supplies them to the LLM.

trial documents uploaded by the user. In this way, the user can search for relevant information in documents such as the CTP, the Trial Statistical Analysis Plan (TSAP), the ADS plan and the Table of Contents (including detailed information about the TLF elements that the programmer needs to produce). Access to the local context via the context API is implemented via the `ellmer` [8] tool logic: we implemented different tools that allow the LLM powering `meRlin` to read files, extract metadata from dataframes and perform dense retrieval according to the Retrieval Augmented Generation (RAG) framework [9].

The second important source of context for the LLM is provided by the *Code Template API* in Figure 1. This service is run remotely on a company internal cloud and allows the LLM powering `meRlin` to access data relevant to the generation of TLF code templates. In particular, the Code Template API consists of a database that is regularly synchronized to three main repositories of code templates:

1. The TLG catalogue [10],
2. The Cardinal template catalogue [11],
3. A company internal template catalogue.

The database indexes the R code for the templates in the catalogues, along with a description of the corresponding TLF element. Then, the user is able to search in the database via a keyword-based method. Again, the search functionality is implemented as an `ellmer` tool that provides context to the LLM.

Finally, we would like to provide some information about the LLM that are used to power the AI agent. The user of `meRlin` can use several proprietary and open weights LLMs, thanks to a custom integration to the `ellmer` package. Because the use of confidential data with public services is prohibited by company policy, we integrate LLMs via an API platform that has passed company approval. This platform allows to select several models from third party providers as well as hosted in the company compute cluster. Users can select models depending on the requirements of their task, favoring smaller language models for simple tasks and larger models for more complex

programming endeavors.
To generate drafts for user required clinical reporting tables, `meRlin`:

1. Identifies available datasets and variables in the R environment.
2. Retrieves relevant table templates from a structured catalogue.
3. Extracts study-specific logic from protocol documents.
4. Generates R code using packages like `rtables`, `tern`, and `cardinal`.
5. Refines and formats the code to match table shell specifications through iterative user interactions.

In the next section we will discuss the current user interface and show examples for reference clinical reporting tables.

RESULTS

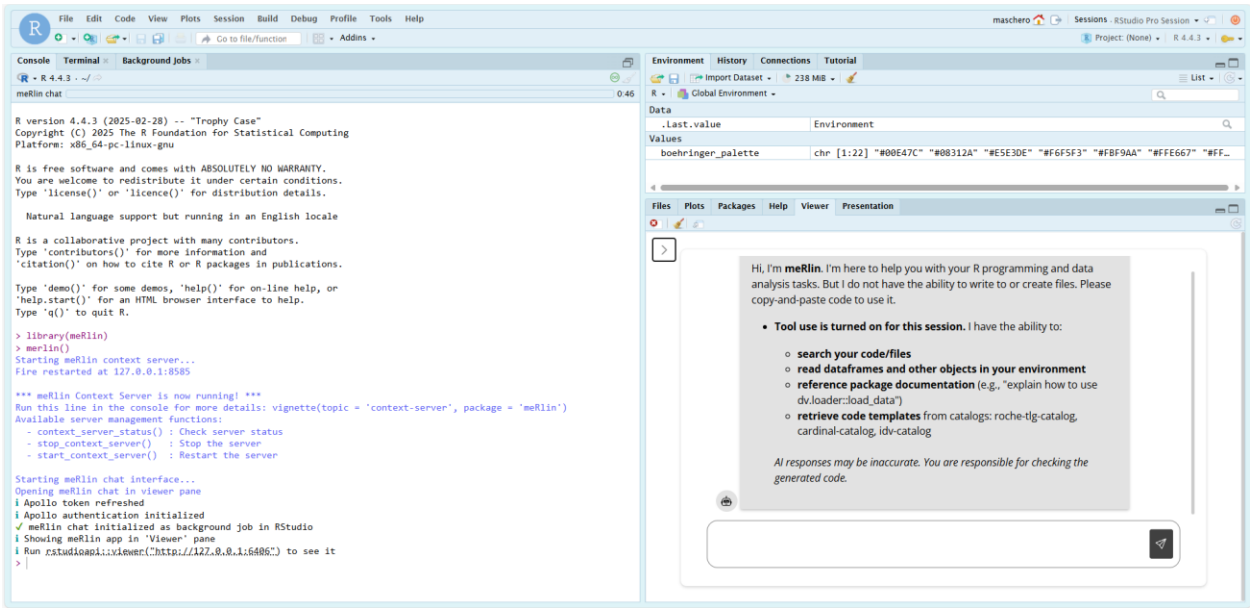

Figure 2: Chat interface for `meRlin`. The user can interact with the agent via the Viewer panel in RStudio, while keeping using the R console.

We begin this section by displaying the user interface of `meRlin`. We first load the package, then start the agent chat interface. The latter is visualized in the Viewer panel of RStudio, as shown in Figure 2. Note that the chat interface can also be started as a separate window, leaving the Viewer panel available for visualizations generated by the R console. By clicking on the arrow at the top left corner, the user can view the Chat History, which records all previous interactions with `meRlin`. Also, the user can select which LLM is powering the agent, choosing from both open weights and proprietary models.
We start showcasing the possibilities offered by `meRlin` by presenting a read operation on a dataframe. We load a dummy ADSL dataset from the `random.cdisc.data` [12] package and ask `meRlin` to comment on its content (see Figure 3). Note that in this example and in the following ones we use the GPT-5 model from OpenAI [13] as the LLM powering the agent. `meRlin` provides a concise summary of the data, displaying the structure of the dataframe and listing the main variables and flags.
As a next example, we show the agent response when being asked to display the R code to generate a clinical table. Figure 4 shows how `meRlin` searches the cardinal repository, fetches the requested table type, and returns a code snippet to generate it in the user R console. The same request can be done without providing a specific repository: `meRlin` will take care of fetching results from all the available templates and will display results to the user, asking for next steps.
Finally, we show how it is possible to combine the agent capabilities to perform a complex task. Figure 5 shows an example in which `meRlin` is tasked to create a demographic table from another sample ADSL dataset. This time the user requires the agent to filter patients before generating the table, specifying conditions that are to be retrieved from the ADS plan. The agent breaks down the query into multiple steps, by fetching the relevant template and searching the ADS plan for flags to use to filter the variables. Then, `meRlin` aggregates the responses of the different tools in a single R script with the instructions to generate the table adhering to the specified conditions.

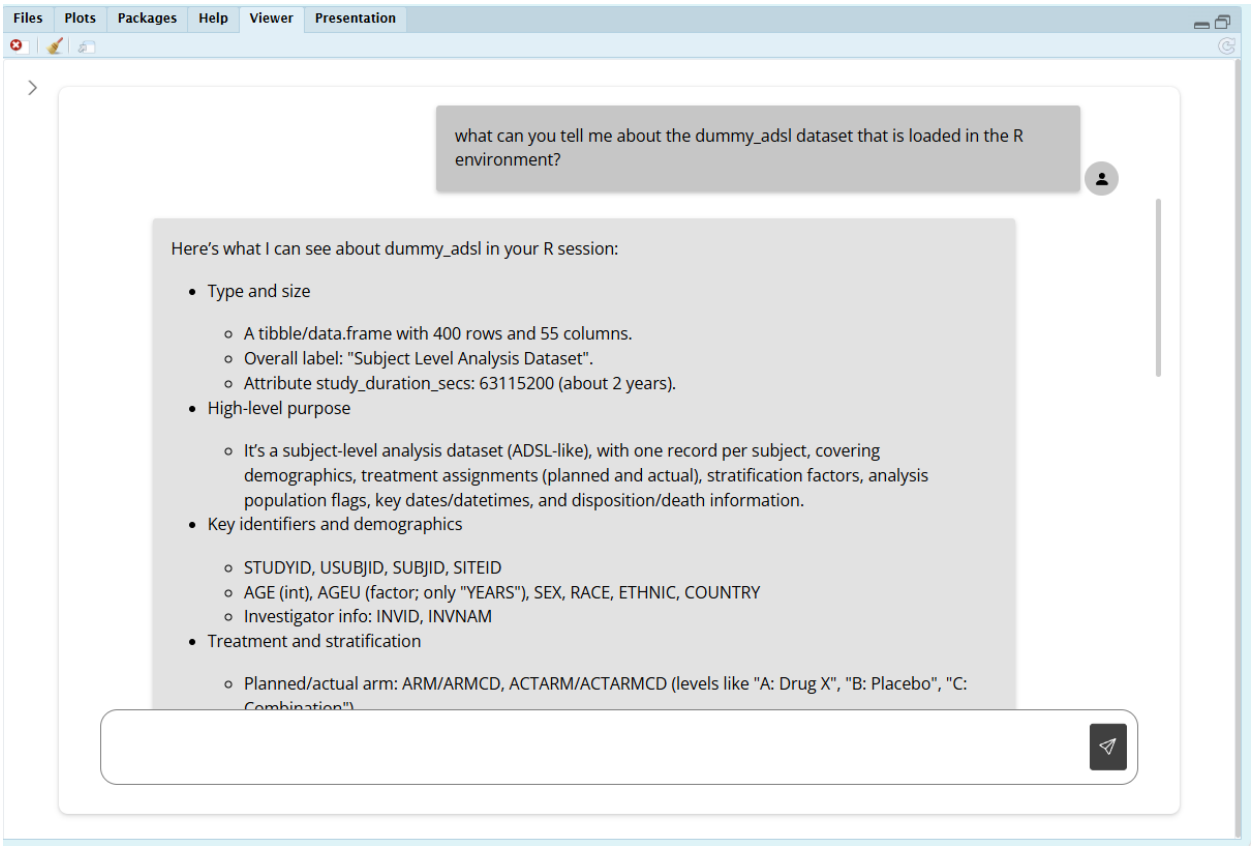

Figure 3: Response of `meRlin` after being asked to comment on a dataframe that is loaded in the R environment.

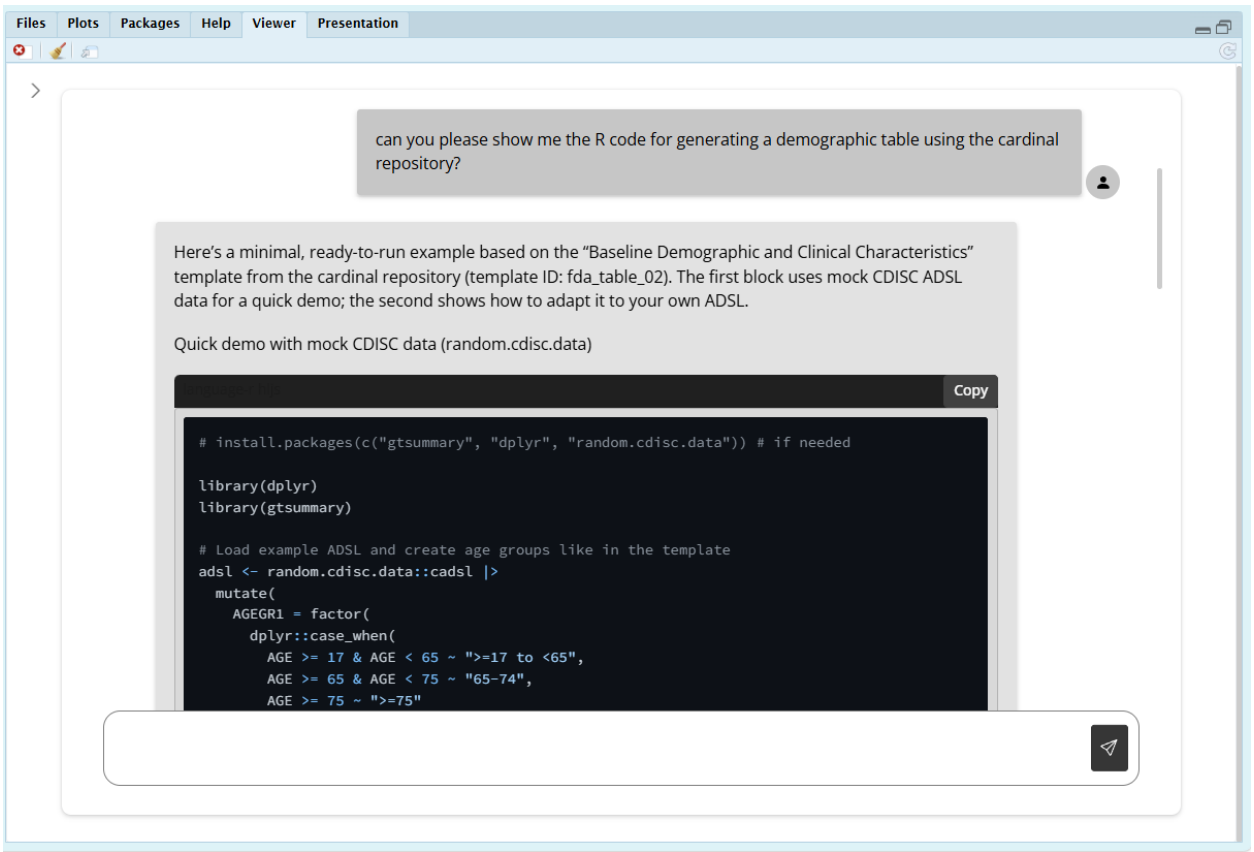

Figure 4: When asked to display the code to generate a demographic table, meRlin fetches the corresponding template from the requested repository and returns it to the user.

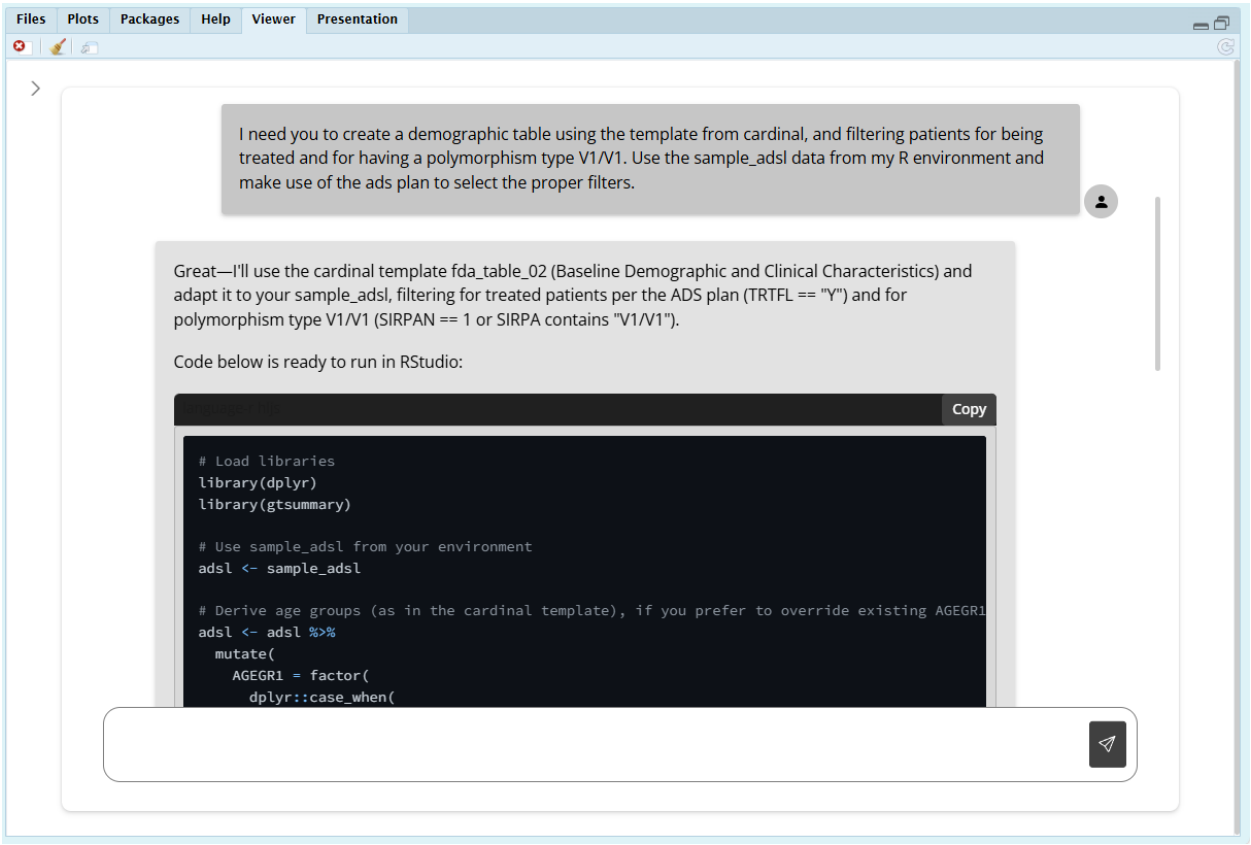

Figure 5: When asked to perform a complex task, `meRlin` uses its set of tools to break it down in multiple steps, fetches the relevant context from local and remote APIs, and then aggregates results to provide an answer to the user.

CONCLUSIONS

In this paper we showcased the development of `meRlin`, an AI agent for supporting statistical programmers and clinical data scientists with producing TLF outputs, in particular clinical reporting tables. The agent searches a database for R code templates that are relevant to the user query, retrieves specific knowledge from trial documents and produces a draft for the R code that will be used to generate the required table. This flow is implemented to be as seamless as possible for the end user, who can query the tool in natural language and work on improving the draft that is generated directly in an RStudio instance. By giving unified access to different (internal and external) template repositories and trial documents, `meRlin` facilitates the workflow of users, with the final aim of increasing the speed and accuracy of the generated TLF outputs.

As for the next steps, we plan to extend the template repositories to cover additional TLF element types (e.g., figures). Also, we are working to adapt the connection of the existing tools to the MCP standard [14], to allow for possible reuse by other agentic applications. Finally, we will continuously add support for new LLMs as they become available, so that users can benefit from technological advancements.

REFERENCES

[1] Lee, Jeong-Moo. "Strategies for integrating ChatGPT and generative AI into clinical studies." *Blood research* 59.1 (2024): 45.

[2] Yang, Yumeng, et al. "Using Large Language Models to Generate Clinical Trial Tables and Figures." *arXiv preprint arXiv:2409.12046* (2024).

[3] Husein, Rasha Ahmad, Hala Aburajouh, and Cagatay Catal. "Large language models for code completion: A systematic literature review." *Computer Standards & Interfaces* 92 (2025): 103917.

[4] Brown, Tom, et al. "Language models are few-shot learners." *Advances in neural information processing systems* 33 (2020): 1877-1901.

[5] Chang W, Cheng J, Allaire J, Sievert C, Schloerke B, Xie Y, Allen J, McPherson J, Dipert A, Borges B (2025). _shiny: Web Application Framework for R_, <https://CRAN.R-project.org/package=shiny>.

[6] Nivard M, Wade J, Calderon S (2025). _gptstudio: Use Large Language Models Directly in your Development Environment_, <https://github.com/stevegbrooks/gptstudio>.

[7] Pedersen T (2024). _fiery: A Lightweight and Flexible Web Framework_, <https://fiery.data-imaginist.com>.

[8] Wickham H, Cheng J, Jacobs A, Aden-Buie G, Schloerke B (2025). _ellmer: Chat with Large Language

Models_, <https://ellmer.tidyverse.org>.
[9] Gao, Yunfan, et al. "Retrieval-augmented generation for large language models: A survey." *arXiv preprint arXiv:2312.10997* 2.1 (2023).
[10] tlg-catalog/book at main · insightsengineering/tlg-catalog · GitHub, accessed on 19.10.2025
[11] GitHub - pharmaverse/cardinal: FDA Safety Tables and Figures, accessed on 19.10.2025
[12] Rucki P, Paszty N, Stoilova J, Zhu J, Garolini D, de la Rua E, DiPietrantonio C, Waddell A (2024). _random.cdisc.data: Create Random ADaM Datasets_.
[13] Introducing GPT-5 | OpenAI, accessed on 19.10.2025
[14] Model Context Protocol · GitHub, accessed on 19.10.2025

DISCLAIMER

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name: Pietro Mascheroni
Company: Boehringer Ingelheim Pharma GmbH & Co. KG
Address: Birkendorfer Str. 65, 88397, Biberach, Germany
Email: pietro.mascheroni@boehringer-ingelheim.com