Paper ML09

Talk to Your Data: Accelerating Insights with AI-Powered Dashboards

Emily Yates, Formation Bio, New York, USA

ABSTRACT

AI has the potential to accelerate data-driven decision making by transforming how we interact with complex datasets. In this session, we introduce an AI-powered RShiny dashboard built using emerging R packages including {ellmer}, {shinychat}, and {querychat}. This embedded chatbot enables non-technical stakeholders to talk to their data - asking natural language questions, exploring patterns, and extracting insights from publicly available datasets without writing a line of code. By integrating large language models into a user-friendly interface, the dashboard democratizes data access and fosters faster, more informed decisions. We'll also explore strategies for selecting the right analytical model and crafting system prompts that result in high quality output for specific use cases. This approach bridges the gap between data scientists and decision-makers, empowering teams to collaborate more effectively and act on insights with confidence.

INTRODUCTION

The increasing complexity of clinical and operational datasets has created barriers to data access and interpretation. Traditional dashboards often rely on pre-defined queries and filters, limiting exploratory analysis and slowing decision-making. AI-augmented dashboards that allows users the opportunity to query datasets using conversational language, eliminating the need for R or SQL expertise.

Recent advances in open-source R packages for large language model (LLM) integration, such as {ellmer}, {shinychat}, and {querychat}, have made it easier to embed generative AI directly into analytical environments. However, these tools can be complex to configure and tune correctly - small changes in prompts, model parameters, or system design can have significant effects on the quality and reliability of the output.

It is therefore critical that AI-augmented dashboards are designed for high quality and reliability, so that automation enhances, rather than interferes with, the user's analytical exploration. The goal is not just to make data more accessible, but to ensure that the AI assists users appropriately and produces outputs that are fit for purpose in a clinical context.

This paper describes how to make the key technical design decisions that most strongly influence the quality of AI-assisted analysis - covering model selection, system prompt design, contextual grounding, and reproducibility. We present a structured framework for evaluating these design choices and highlight best practices for developing trustworthy, high-performance AI-powered dashboards that accelerate insight generation across teams.

SYSTEM DESIGN AND TECHNICAL DECISIONS

The foundation of the AI-augmented dashboard is a streamlined system design that connects user intent with executable code.

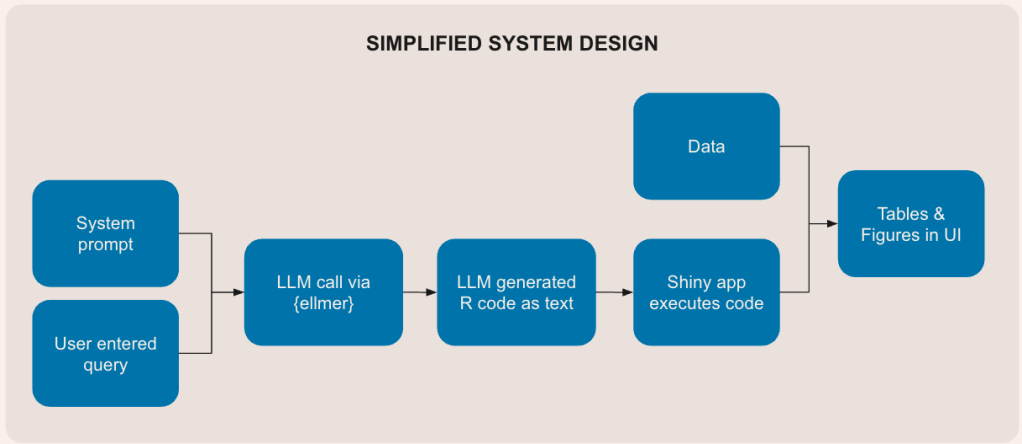

Figure 1: simplified system design for the AI-powered R-Shiny dashboard

The process begins when a user enters a query in natural language. The query is passed, along with a predefined system prompt, to the {ellmer} interface, which connects to a large language model such as GPT-4 or GPT-5. Key technical design decisions that strongly influence the quality of AI-assisted analysis occur in the call to {ellmer}.

The model generates R code as plain text, which the Shiny application executes to produce dynamic tables and visualizations. This architecture allows users to iteratively refine their questions and immediately see updated results - enabling a conversational approach to data exploration. Importantly, this system design does not expose data to the LLM so including sufficient metadata and context in the system prompt is critical to high quality output.

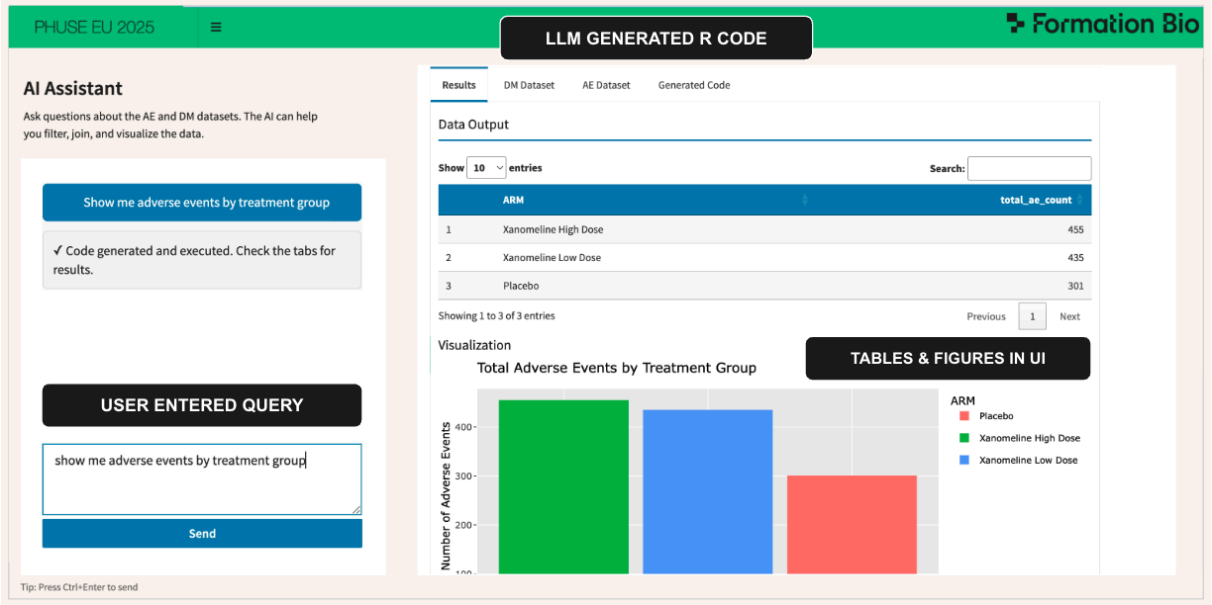

Figure 2: R Shiny UI showing the user entered query & generated table & figure

This dashboard will serve as a point of experimentation to demonstrate the impact and importance of the following technical design decisions. This dashboard has access to test AE and DM datasets from the {pharmaversesdtm} package and serves to answer questions about patient adverse events and safety information.

CHOOSING THE RIGHT MODEL

Model selection is one of the first consequential design choices in developing an AI-augmented dashboard. Each model family offers a different balance of reasoning ability, cost, latency, and reproducibility, and the “right” choice depends on the specific analytical context and organizational constraints.

- **GPT-5** delivers the strongest reasoning, long-context handling, and code-generation capabilities. It excels at solving complex, ambiguous queries but comes with higher computational cost and latency.
- **GPT-4.1** provides high-quality reasoning and reliable code generation at a lower price point. It maintains consistency across varied tasks and offers an excellent balance between performance, stability, and speed.
- **GPT-4.0** remains a proven and reliable option for deterministic use cases, especially when stability and consistency are more important than cutting-edge reasoning.
- **GPT-5-mini** and **GPT-4.0-mini** are optimized for speed and cost-efficiency, making them ideal for structured, repetitive queries or lighter-weight dashboards.

In practice, the goal is to optimize the trade off between performance, latency, and price by selecting the least complex model that consistently produces high quality results. A model that is too powerful for the task can slow performance and increase cost without improving accuracy or user experience. The most suitable model is the one that delivers results that are accurate, interpretable, and aligned with user needs.

For this implementation, GPT 4.1 was selected. It provided the best balance of reasoning quality, stability, and cost efficiency for generating R code within the dashboard. GPT 4.1 consistently delivered accurate and contextually appropriate outputs while maintaining a fast enough response time to support a smooth, interactive experience. This combination made GPT 4.1 the most fit-for-purpose choice for an AI-powered dashboard designed to enable exploratory yet dependable data analysis.

Model	Excels At / Strengths	Trade-offs / Weaknesses	Price (per 1M tokens)
GPT-4.0 mini	Lightweight, fast, cost-efficient; good for simple or high-volume tasks	Limited reasoning; weaker on ambiguous or open-ended prompts	**Input:** $0.15 **Output:** $0.30
GPT-5-mini	Good for structured tasks, high-volume usage, simpler or well-specified queries	Not as strong in edge cases, complex logic, or ambiguous prompts	**Input:** $0.25 **Output:** $2.00
GPT-4.0	Stable, proven model; supports seed for reproducibility; good for deterministic demos	Weaker reasoning than newer models; being gradually deprecated	**Input:** $0.30 **Output:** $0.60
GPT-4.1	Strong reasoning, reliable code generation, balanced speed vs quality	Less powerful than GPT-5; cost higher than 4.0/mini	**Input:** $0.50 **Output:** $1.50
GPT-5	Best for most complex reasoning, code generation, long / mixed context, high-quality output	Higher cost, more resource usage, possibly higher latency	**Input:** $1.25 **Output:** $10

Table 1: Framework to help decide which model is best suited for this use case.

HIERARCHY OF PROMPTS

Large language models process instructions through a hierarchy that determines which inputs take precedence. Understanding this hierarchy helps developers predict how the model will behave and design prompts that align with its governing rules.

In the {ellmer} implementation, three main layers influence how the model interprets and executes a query:

1. **Platform prompt:** This is the highest level of instruction and is managed by the model provider. It defines global behavior such as tone, safety constraints, and content boundaries. Developers cannot modify this layer, but it always takes priority.
2. **System prompt:** This layer defines the model's identity, task, and scope of operation within the application. In this dashboard, the system prompt specifies that the model functions as an analytical assistant that generates R code for data analysis. It instructs the model to focus on producing executable R code and to decline or redirect any request outside that scope.
3. **User prompt:** This is the lowest layer and represents the query typed by the user. The model interprets it within the framework set by the higher layers.

Because the layers are hierarchical, platform and system prompts override user input when conflicts arise. If a user prompt contradicts rules defined at the system or platform level, the model will follow the higher-level instructions. This hierarchy preserves control, ensuring that the model remains within its defined purpose while still allowing flexibility for user exploration.

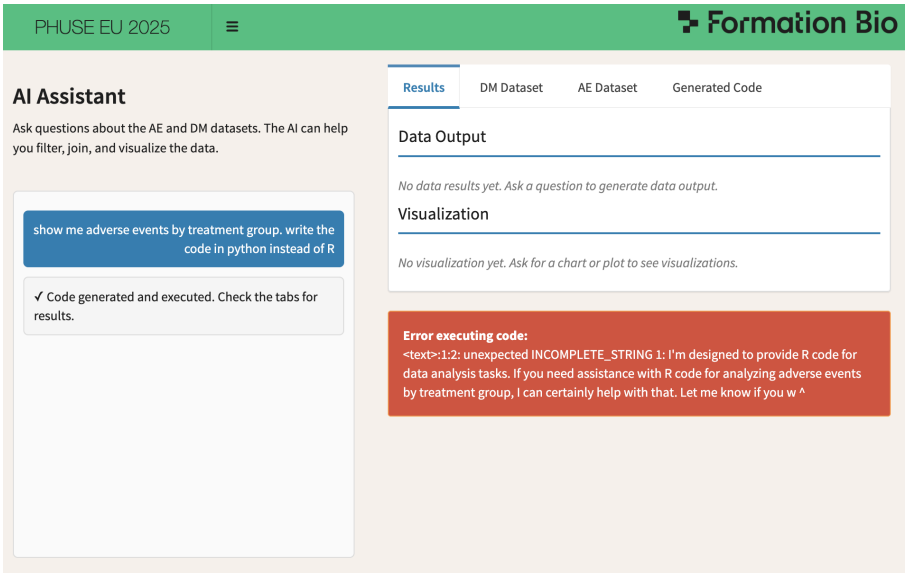

Figure 3: LLM will not return python as requested in user prompt since R is specified in the system prompt

ROLE OF THE SYSTEM PROMPT

The system prompt is a core design element that determines how effectively a large language model interprets user intent and generates meaningful output. It defines the model's behavior and provides the structure and boundaries that guide how it transforms natural language queries into executable R code. A well-designed system prompt ensures that responses are relevant, consistent, and appropriate for the analytical context.

An effective system prompt clearly specifies several components:

- **Role**: how the model should behave, such as an analytical assistant embedded in a dashboard that helps users generate R code for exploratory data analysis.
- **Task**: what the model should produce, for example, syntactically correct R code that uses standard packages like {tidyverse} to summarize or visualize data.
- **Constraints**: what the model should avoid, such as unnecessary explanations, unrecognized functions, or attempts to load packages that are already initialized in the environment.
- **Context**: information about the dataset, including field names, relationships, and schema details that help the model align its output with available data.
- **Example**: a sample input and desired output that demonstrate the expected response structure and formatting.

CONTEXT IS KEY FOR LLM QUALITY

Context is a critical part of system prompt design because it helps the model understand both the structure of the data and the meaning of the analysis. Supplying the right context improves accuracy, reduces unnecessary assumptions, and ensures the generated code aligns with the dataset and analytical goals. In this implementation, two main types of context proved especially important: data context and domain context.

DATA CONTEXT

Data context defines the structure and content of the dataset. It tells the model what variables exist, how they relate to one another, and how they should be used in code. Without explicit data context, the model may invent field names or misunderstand how tables connect. Supplying data context gives the model the technical foundation it needs to generate syntactically valid and logically accurate R code.

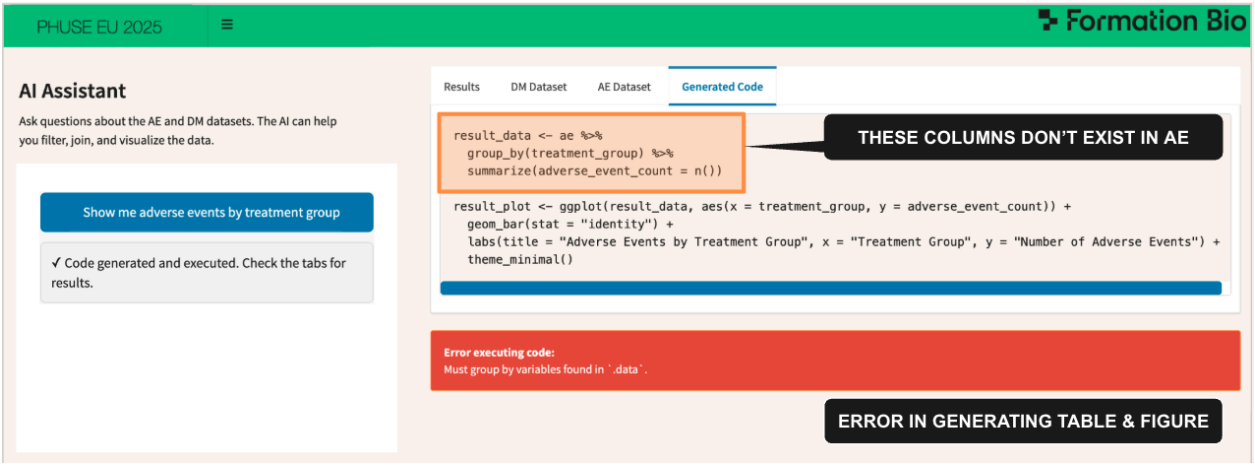

Figure 4: Without proper data context, LLM generates erroneous code & invents variables that don't exist

When LLM-generated R code is syntactically incorrect, produces errors, or shows other signs of not technically understanding the dataset, consider adding data context to the system prompt. This may include clarifying variable names, data types, or table relationships so the model can correctly reference and manipulate the source data.

Examples of data context:

- **Field-level metadata**: Indicating that variables such as usubjid, actarm, and aedecod exist and defining their types (for example, categorical or numeric).
- **Schema relationships**: Explaining how tables like demographics and adverse events link through shared keys such as usubjid.

- **Data dictionaries**: Including descriptions of key fields, such as those found in the cdisc sdtm implementation guide, to clarify variable meaning.
- **Missing data patterns**: Informing the model that certain variables may be incomplete or recorded at irregular intervals, which guides appropriate handling in summaries or visualizations.

Providing this level of detail helps the model produce syntactically correct code that accurately reflects the data structure and avoids unnecessary corrections by the user.

DOMAIN CONTEXT

Domain context provides the scientific or operational meaning that gives analytical results real-world relevance. Even technically correct code can produce analyses that are misleading or incomplete if the model does not understand the clinical or scientific concepts represented in the data. Supplying domain context ensures that the generated analyses are meaningful, interpretable, and aligned with how experts understand the subject matter.

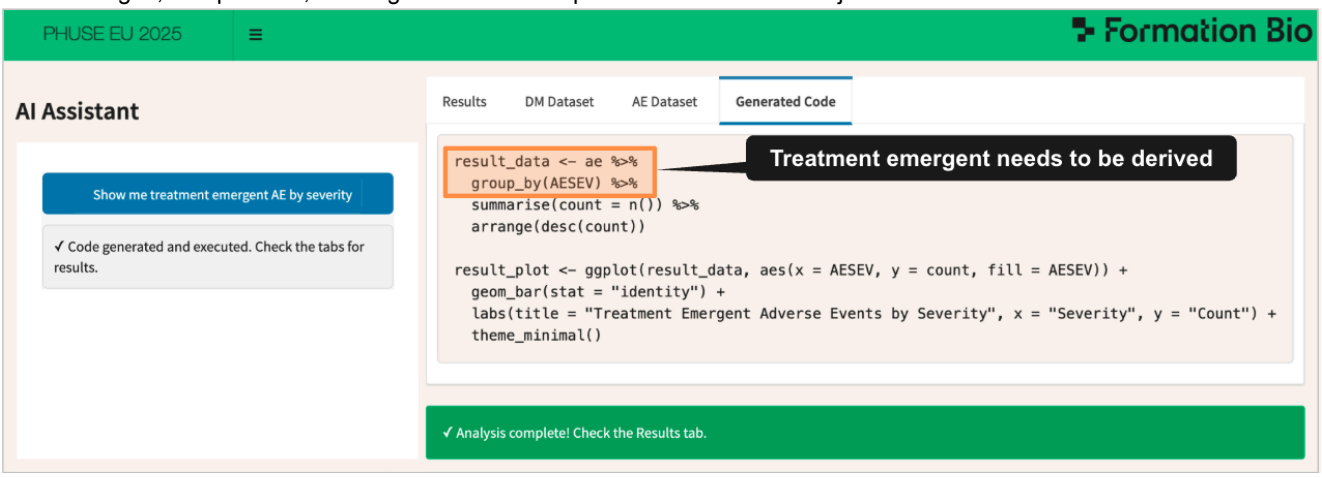

Figure 5: Without proper domain context, code can execute without error but the analysis is incorrect

A model may need additional domain context when outputs are technically correct but conceptually off. Common signs include analyses that ignore study structure, use incorrect terminology, or produce results that do not make sense in the clinical setting. If outputs seem generic or inconsistent with expected logic, the prompt likely needs more domain-specific information about variables, definitions, or analysis goals.

Examples of domain context:

- **Semantic mapping**: Linking clinical concepts to their technical derivations. For example, defining how to derive a safety population or treatment-emergent adverse events (TEAEs) from raw datasets.
- **Domain ontologies**: Using established frameworks such as MedDRA or CDISC to guide terminology and ensure consistent interpretation across datasets.
- **Subject matter expertise**: Incorporating short notes or definitions from clinical scientists or statisticians that clarify the intent of key fields or expected relationships.
- **Concept harmonization**: Aligning similar variables across datasets

Providing domain context helps the model generate analyses that not only run correctly but also make sense in the clinical or operational setting where they are used.

IMPORTANCE OF ITERATION

System prompt design is not a one-time exercise. Because large language models interpret instructions probabilistically, small adjustments to the prompt can lead to noticeable differences in output quality. Iterating on the prompt allows developers to refine how the model interprets intent, handles context, and generates R code that meets the analytical objective. LLMs can also be a helpful tool in identifying where the prompt may be lacking by analyzing its own errors or misinterpretations, offering clues about missing context, unclear instructions, or ambiguous language that should be clarified in future iterations.

TEMPERATURE AND OUTPUT CONSISTENCY

Temperature controls how a large language model generates text. It determines how deterministic or variable the model's responses will be. Lower temperatures make outputs more consistent, while higher temperatures encourage more creative or exploratory behavior.

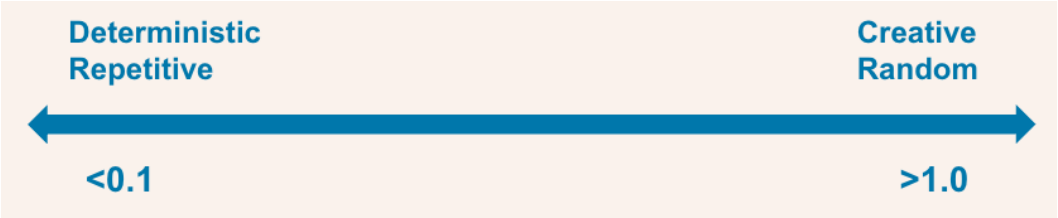

Figure 6: Range and impact of temperature values

At low temperatures (typically between 0.0 and 0.3), the model prioritizes the most likely next token in a sequence, resulting in predictable, stable outputs. This setting is preferred for generating code or performing structured analytical tasks, where reliability and precision are more important than creativity.

At high temperatures (typically between 0.7 and 1.0), the model introduces more randomness into its responses, making it more willing to explore alternate phrasings or unconventional solutions. This can be useful for brainstorming or exploratory analysis but often results in inconsistent or syntactically incorrect R code.

In practice, temperature should be tuned based on the desired balance between creativity and consistency. For an AI-augmented dashboard, a low temperature setting is typically ideal because users expect reproducible analytical outputs. A higher temperature can be useful during testing or development to observe a broader range of model behaviors, but production environments benefit from predictability.

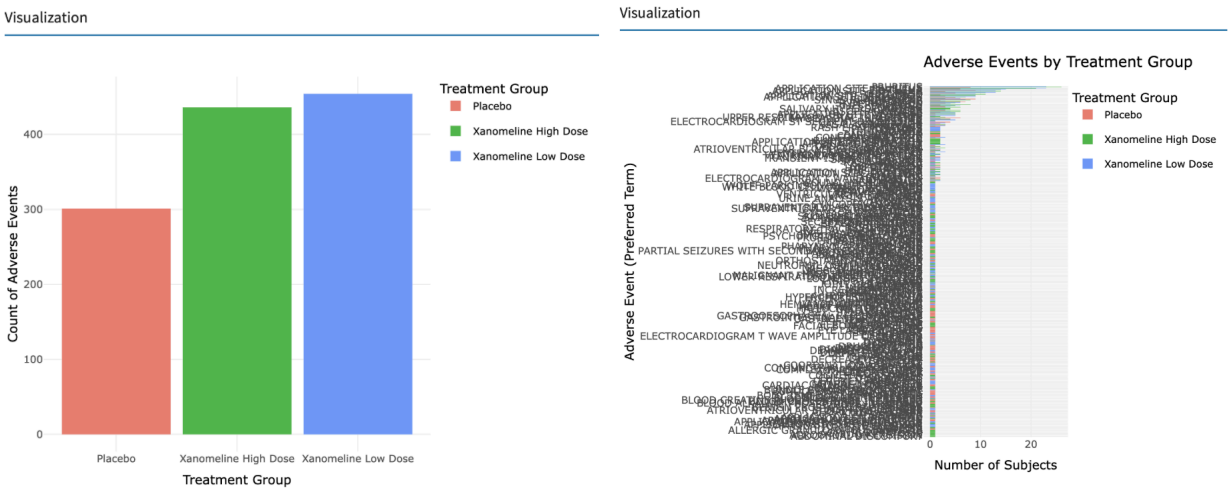

Figure 7: Low temperature (left) versus high temperature (right) output showing impact of randomness

SEED AND CONTROLLED RANDOMNESS

The seed parameter initializes the model's random number generator, allowing developers to control how much variation occurs between runs. While temperature determines the degree of randomness, the seed determines whether that randomness is repeatable.

When a seed is defined, the model's outputs will follow the same probabilistic path for identical inputs, producing consistent results each time the same prompt is executed. When no seed is defined, the model samples freely from the probability space, which can lead to subtle variations even at the same temperature setting.

Using a fixed seed is valuable when evaluating prompt quality or demonstrating results, since it allows developers to compare outputs across iterations under identical conditions. For example, setting a seed in the {ellmer} call ensures that the same R code and corresponding figures are generated each time the user submits a specific query.

However, a seed does not remove randomness altogether; it simply makes that randomness predictable. Changing the seed value produces a new, but internally consistent, variation in output. This can be useful for exploring how the model interprets different phrasing or for testing robustness to small perturbations in input.

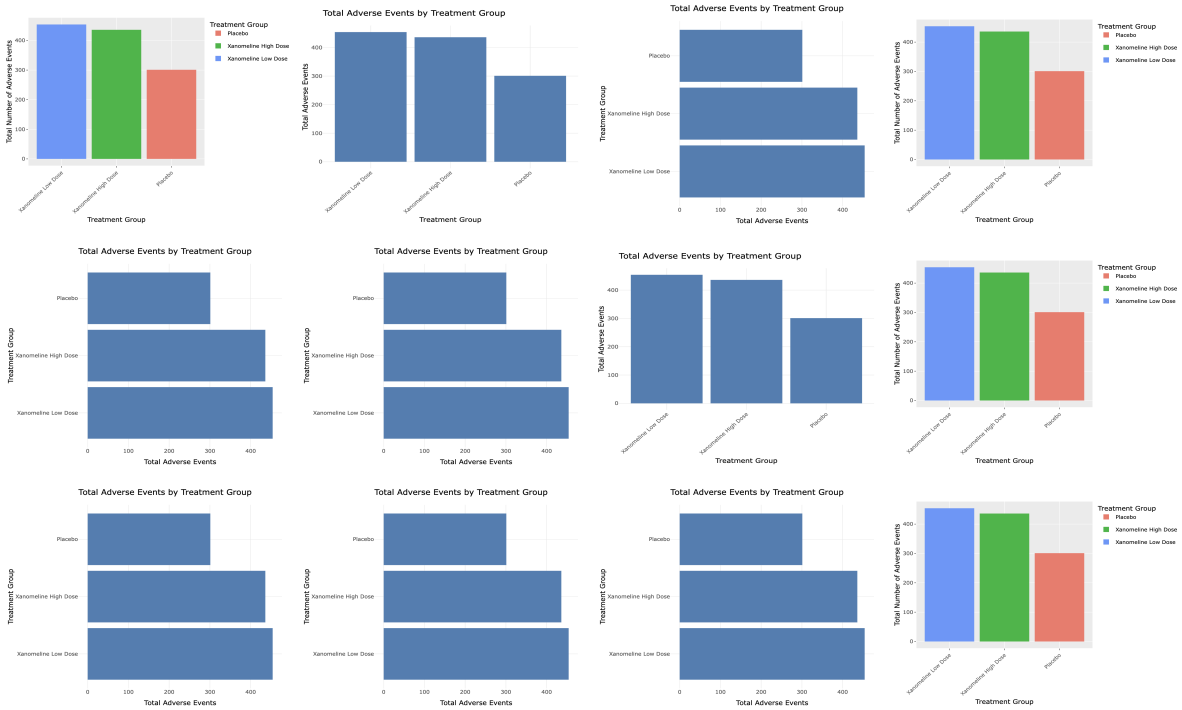

Figure 8: Without a seed, there is more variation in the types of visualizations produced from the same user prompt

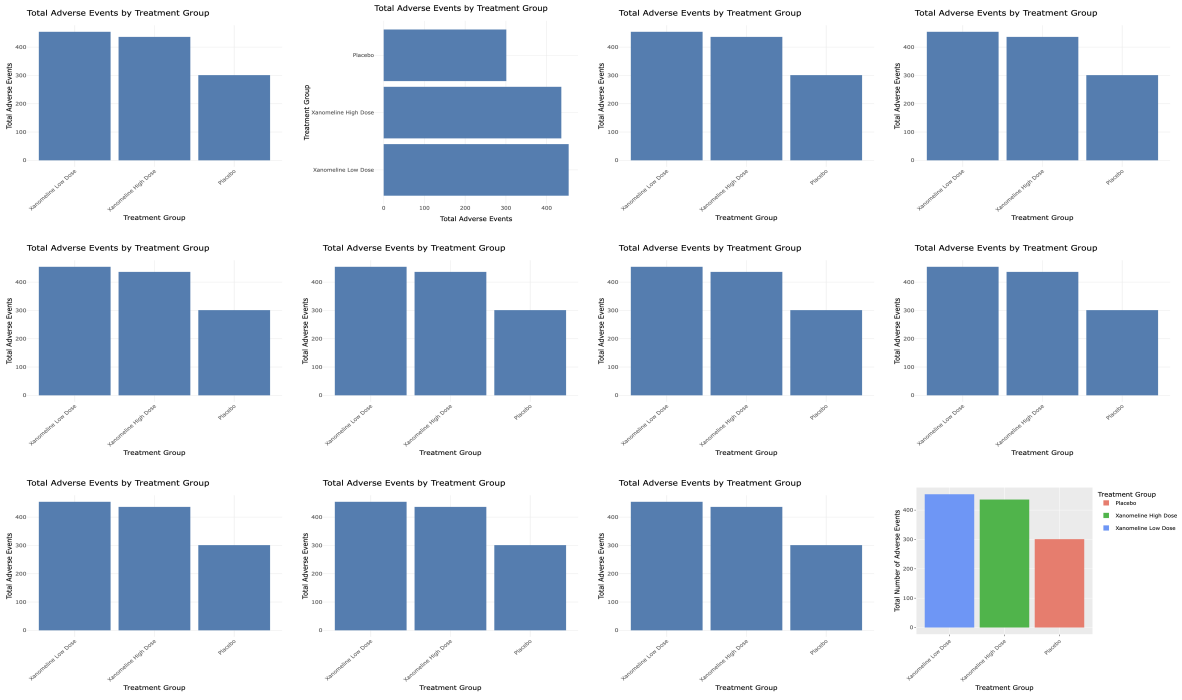

Figure 9: With a seed, there is less variation in the types of visualizations produced from the same user prompt

CONCLUSION

Designing an AI-augmented dashboard requires balancing capability, control, and clarity. The goal is not simply to embed an LLM, but to ensure that the tool enhances analysis without introducing unnecessary complexity or inconsistency.

Model selection should focus on using the least complex model that consistently produces high-quality results. More advanced models do not always lead to better outputs. The right choice depends on the analytical use case, data sensitivity, and user needs.

System prompt design benefits from iteration and context. Refining the prompt over time improves how the model interprets intent and generates code. Supplying both data context and domain context ensures that outputs are technically accurate and clinically meaningful.

Temperature and seed parameters work together to control variability and determinism. A lower temperature combined with a defined seed produces stable, reliable responses, while higher settings can be used during development to explore a broader range of model behaviors.

Together, these principles form a framework for building AI-powered dashboards that are transparent, dependable, and fit for purpose. As these tools continue to evolve, thoughtful technical design - grounded in experimentation, context, and constraint - will be the key to turning generative AI from a novelty into a practical asset for data-driven decision-making.

REFERENCES

Yates, E. (2025). *Maximizing LLM Potential- How Data Structure and Context Drive Quality Insights* . Presentation slides. Available at: https://github.com/emurphy-TS/RinPharma-GenAI-2025/blob/main/Maximizing%20LLM%20Potential-%20%20How%20Data%20Structure%20and%20Context%20Drive%20Quality%20Insights%20%20.pdf

CDISC. *Study Data Tabulation Model (SDTM) Implementation Guide*. Available at: https://www.cdisc.org

{pharmaversesdtm} R package. Pharmaverse. Available at: https://pharmaverse.github.io/pharmaversesdtm

{ellmer} R package. Posit Software. Available at: https://github.com/posit-dev/ellmer

{shinychat} R package. Posit Software. Available at: https://github.com/posit-dev/shinychat

{querychat} R package. Posit Software. Available at: https://github.com/posit-dev/querychat

CONTACT INFORMATION

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

Emily Yates
Formation Bio
eyates@formation.bio