Paper TT08

Blockr: Advancing Data Pipelines with DAG-Based Workflows and LLM Orchestration

Karma Tarap, Bristol Myers Squibb, Boudry, Switzerland

Nicolas Bennett, Cynkra, Zurich, Switzerland

ABSTRACT

The blockr ecosystem is an open-source framework designed to simplify data analysis and dashboard creation. Originally presented at PHUSE 2024 as a block-based dashboard builder, it has since evolved into a modular and extensible ecosystem that supports the definition, execution, and validation of reproducible analytical workflows. This paper provides an overview of its system architecture, extensibility model, and applications in clinical research. It also discusses lessons learned from its development and deployment at Bristol Myers Squibb (BMS).

BACKGROUND AND MOTIVATION

As data volumes and regulatory requirements increase, clinical analytics teams face growing complexity in managing workflows. Traditional Shiny dashboards often become difficult to maintain, reproduce, or validate. Analysts and statisticians frequently rely on programming expertise that may not be available to all users, leading to delays and siloed analyses. The goal of blockr is to reduce these barriers by providing a flexible, visual interface that translates analytical design directly into reproducible R code.

The initial version of blockr introduced a visual programming paradigm for R, where analytical components—called blocks—could be assembled into data pipelines. Over the past year, the project expanded into a multi-package ecosystem that separates core logic, user interface components, and analytical extensions. This architecture improves maintainability and scalability while allowing different teams to contribute specialized blocks for their analytical needs.

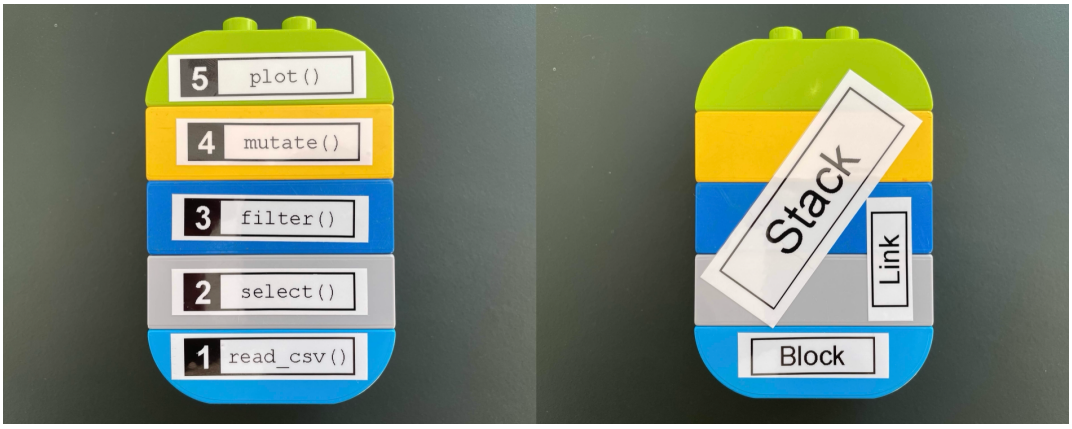

Figure 1. Blocks assembled into a stack provide composable units for building analytical pipelines.

CORE CONCEPTS

The blockr ecosystem is structured around four primary concepts: blocks, links, stacks, and boards. Data operations—such as import, transformation, or visualization—are provided by blocks. Each implements functionality as modular R expressions. The data flow between blocks is coordinated by links, while stacks contribute a grouping mechanism to organize blocks into disjoint sets. The directed acyclic graph (DAG) of connected blocks forming analytical workflows and organized via stack membership is a board. Boards therefore serve as workspaces for managing analysis projects.

Each block defines its inputs, outputs, and execution logic using a declarative syntax. Both a single block (for development purposes), as well as an entire board can be used to start up a Shiny application. This guarantees that every blockr analysis, regardless of how it was built, can be validated and rerun independently.

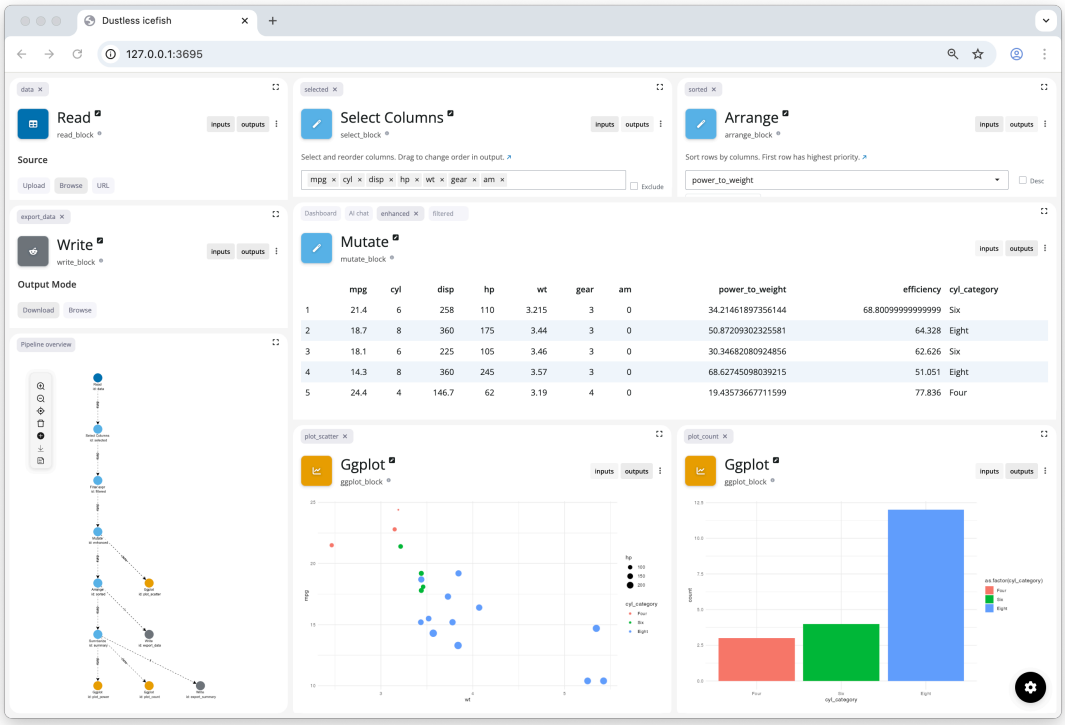

Figure 2. Interactive blockr workspace showing linked analytical components.

EXTENSIBILITY

A key strength of the blockr ecosystem lies in its extensibility. Two APIs support customization:

1. The Block API allows developers to define analytical primitives by specifying inputs, outputs, and execution logic.

2. The UI Extension API enables customization of the interface, such as adding project management, UX customization and code export via plugins, as well as extensions such as AI assistants or document builders.

Both APIs allow component implementers to build on tried and tested Shiny tools for specifying UI and server logic. Implementations of such components are available in several R packages outlined below, but the APIs are designed for third-party extensibility as well.

Block API

Blocks are defined declaratively, enabling straightforward registration and reuse. Blocks combine data inputs, user inputs and code, which together generate an output. A block implementer is responsible for building out a simple user interface that captures user inputs and interpolates an expression with input values. This expression can then be evaluated in the context of data inputs (results from upstream blocks) to produce a reproducible output.

For example a block which implements column sub-setting on tabular input data could be implemented with a single user input element that provides a drop down with column names (populated, based on the columns detected for the input dataset). The user selection is then spliced into an expression template that encodes the sub-setting operation and evaluated using the input data to produce an output dataset containing only the selected columns.

At BMS, this approach allows developers to publish validated analysis blocks (e.g., standard efficacy or safety computations) that can be reused across studies, promoting efficiency and consistency.

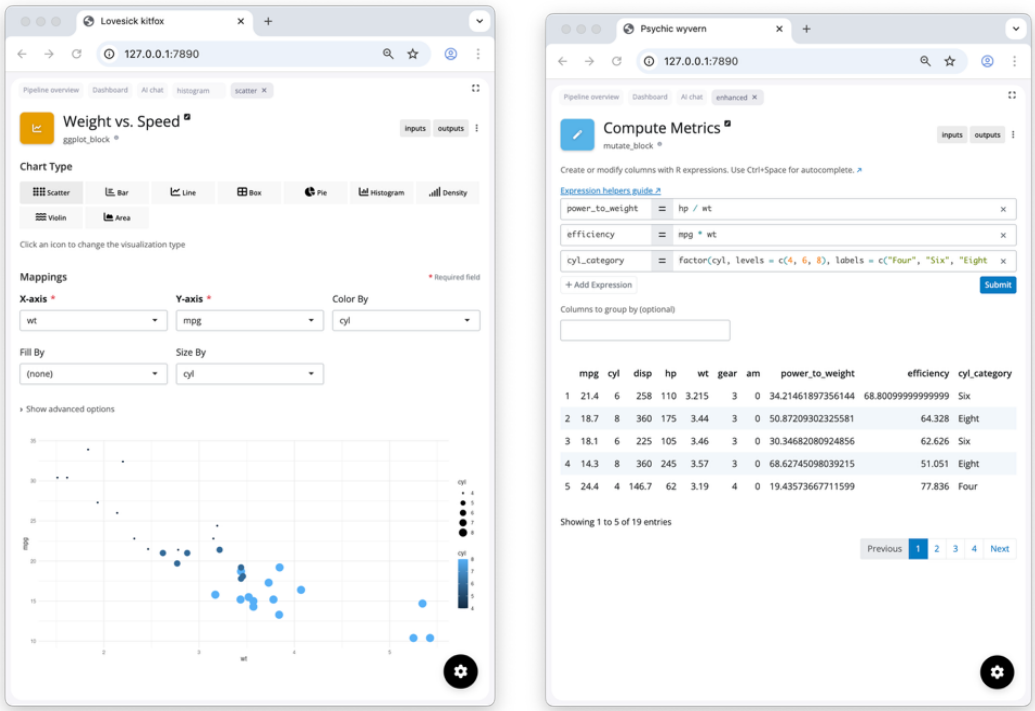

Figure 3. Examples of blocks that implement ggplot2 (left) and dplyr (right) functionality.

User-defined blocks can be registered for use in an application and are best provided in the form of R packages, where a single package might bundle several (related) blocks. The core package contains some blocks itself, but most are more for illustration purposes and unit testing. More user friendly blocks for data manipulation (blockr.dplyr), data visualization (blockr.ggplot) and I/O operations (blockr.io), as well as LLM-powered AI blocks (blockr.ai) are provided by the respective packages. Further functionality for database operations (blockr.db) and table-building (blockr.gt) is being developed, but currently is in an experimental state.

UI/UX Extension API

The core package defines several extension points for certain well-defined pieces of functionality. This includes serialization/de-serialization, management of user notifications and code generation/export, as well as manipulation of board state (adding and removing blocks, links and stacks). We call these plugins and the core package provides basic implementations, which can then be swapped for more capable replacements.

One such example is serialization/de-serialization, where the core plugin offers download and upload buttons which can be used to export/import JSON-serialized data, allowing to save and restore a board. This plugin can be replaced by an alternative implementation offered by block.session, which allows for saving to and restoring from various storage back-ends, such as Posit Connect “pins”, S3 buckets or server-accessible file-systems. Extending this to include database systems is planned but has not been implemented so far.

A front end implementing a docking layout manager (based on the dockview library) is available from blockr.dock, which allows for setting up a grid of panels that can freely be rearranged by the user. Panels can either display block inputs/outputs or extension user interfaces. Several such extensions have been developed so far, including

- *blockr.dag*: a graphical visualization of the DAG underlying the analysis with a point and click user interface to manipulate the DAG via actions such as adding/removing/appending blocks, linking blocks together, grouping blocks into stacks, etc.,
- *blockr.assistant*: an AI assistant (in-development) which can turn natural language into board manipulations (i.e. the assistant can be instructed to add, remove, group, link blocks and therefore set up entire workflows simply by explaining intent),
- *blockr.md*: a markdown document builder, which can integrate block outputs into markdown which can then be rendered to various output formats via pandoc.
- *blockr.ai*: LLM Coding Agent Blocks, which embed LLM-generated code execution within blockr’s reproducible pipeline framework. These blocks serialize both prompts and generated code, enabling deterministic re-execution and full auditability

The currently available extensions serve to highlight the immense flexibility that comes with our approach. Both the user experience for manipulating board state, as well as how the block outputs are assembled into a report (this being a document, a dashboard or something else entirely) is extensible and customizable.

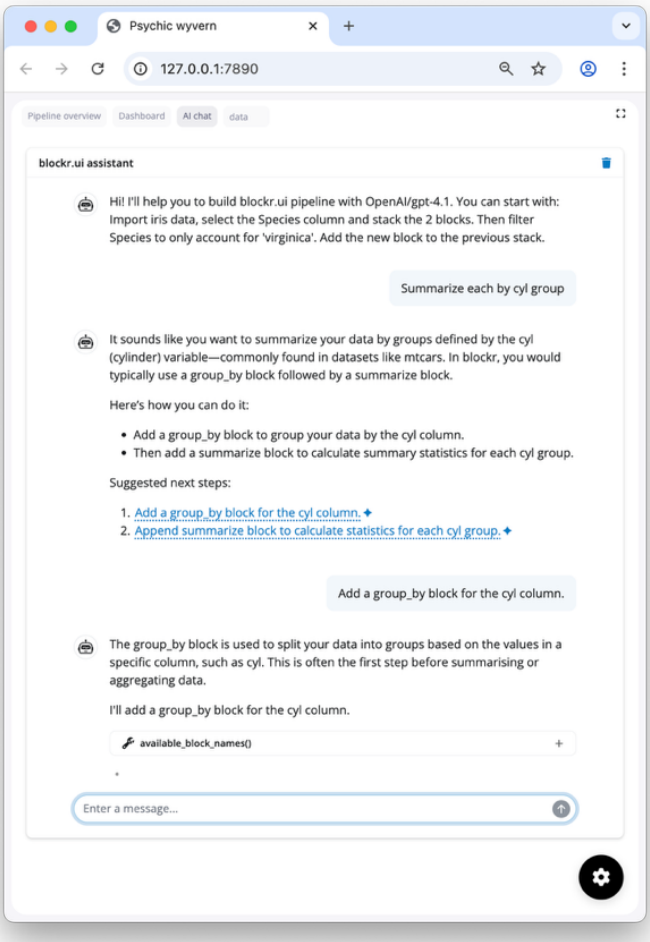

Figure 4. Example of the AI assistant extension provided by blockr.assistant.

APPLICATIONS IN CLINICAL RESEARCH

The blockr ecosystem is currently applied within clinical programming workflows at BMS to support exploratory and standardized analyses. Its key advantages include reproducibility, interactivity, and integration with existing validation pipelines. Clinicians and statisticians can build dashboards and analyses visually, while the system ensures traceability by exporting R code that exactly matches the executed pipeline.

For example, topline reports contain the analysis of clinical trial outcomes for internal decision-making. This typically involves an extensive power point presentation that is generated from RTF data.

A typical blockr topline workflow includes, RTF Data Import, Data Transformation, Interactive Visualization and combining multiple analysis blocks into a comprehensive Powerpoint report.

The blockr.topline package demonstrates how domain-specific workflow can built interactively in the UI, saved as "pins" for reuse and exported as reproducible R code. Some domain specific block are provided by blockr.topline that allow for parsing RTF files, and the rest of the required functionality comes from

components outlines above, including the markdown document builder from blockr.md. The project is still in the development phase but we expect this approach to reduce topline report creation time substantially.

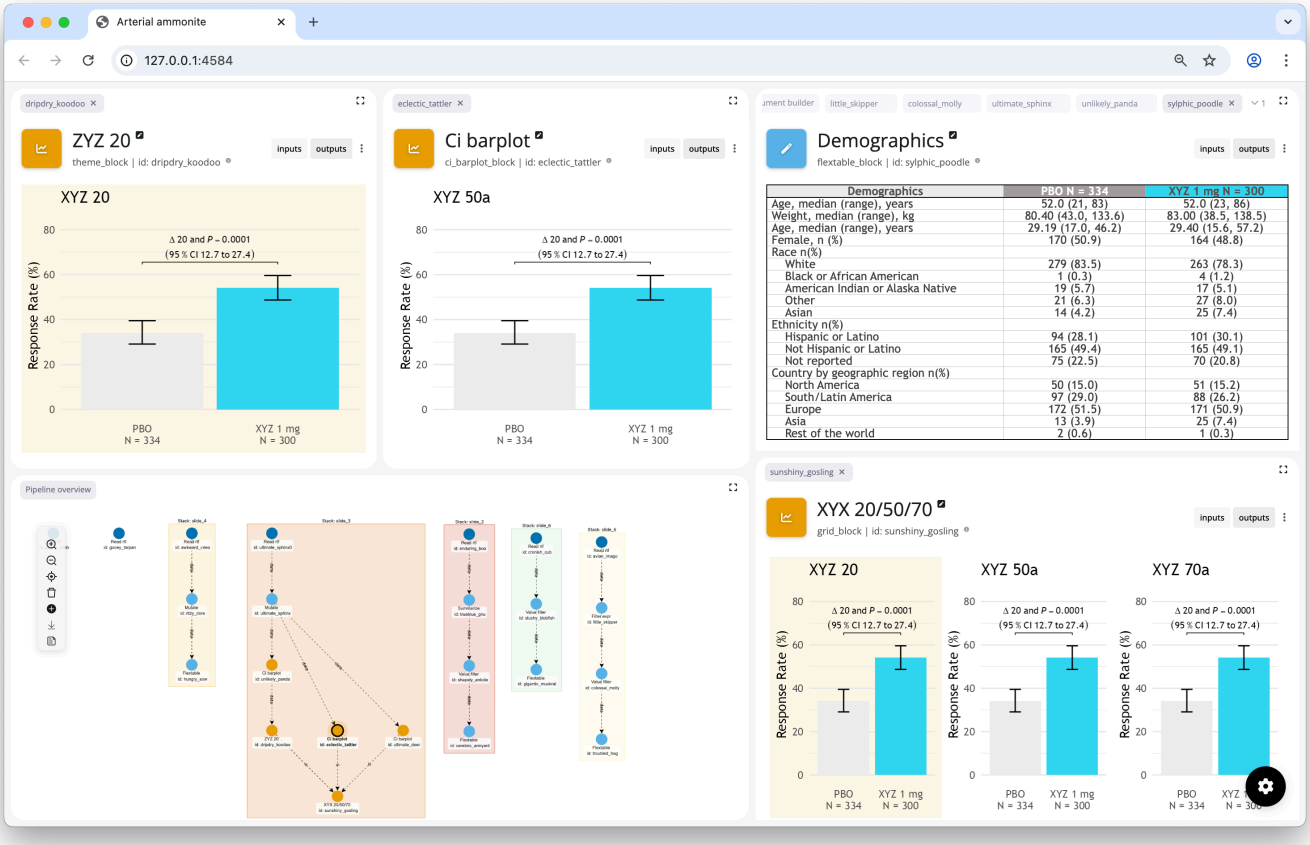

Figure 5. Example of a topline report being built with blocks provided by blockr.topline.

COMPARATIVE ANALYSIS

Compared to commercial platforms such as Power BI ®, JMP Clinical ® or Spotfire ®, blockr offers an open, extensible alternative that minimizes licensing costs. While commercial tools provide mature interfaces and vendor support, blockr emphasizes flexibility and integration with existing R-based workflows. This makes it particularly suitable for organizations seeking to standardize analytical frameworks across studies without vendor lock-in.

CONCLUSION

The blockr ecosystem demonstrates how open-source innovation can coexist with the demands of regulated analytics. By combining modular design, visual composition, and reproducible code generation, it

lowers barriers for analysts while maintaining full traceability and auditability. Its architecture encourages reuse, validation, and collaboration—key ingredients for scalable, transparent analytics in clinical research.

With the integration of LLM-assisted development and code-generation agents, blockr now extends its reach from visual analytics into intelligent workflow authoring and AI-augmented computation. Alongside these innovations, features such as the markdown builder further demonstrate the platform's versatility—enabling automated generation of topline slides and reports directly from validated clinical outputs.

Blockr continues to evolve from a dashboard builder into a comprehensive ecosystem for reproducible, intelligent, and communicative analytical workflows, demonstrating how flexible, open frameworks can meet enterprise-grade standards of compliance, traceability, and scientific quality.

REFERENCES

1. Tarap, K., Coene, J., Bennet, N., Granjon, D., & Sax, C. (2024). *Block by Block: Introducing blockr, the flexible open-source dashboard builder* (Paper OS12). Presented at the PHUSE European Connect 2024. Retrieved from https://www.lexjansen.com/phuse/2024/os/PAP_OS12.pdf
2. Bristol Myers Squibb. (2025). *blockr ecosystem* [GitHub repository]. https://github.com/BristolMyersSquibb/blockr
3. Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis.* Springer.
4. Wilkinson, L. (2005). *The grammar of graphics.* Springer.
5. Pharmaverse. (2024). *Cardinal: FDA safety tables and figures.* https://pharmaverse.github.io/cardinal/

ACKNOWLEDGEMENTS

This work represents a collaborative effort between cynkra GmbH, The Y Company, and Bristol Myers Squibb.

The project was jointly developed by Nicolas Bennet (cynkra) and David Granjon (cynkra), with key contributions from John Coene (The Y Company), Karma Tarap (Bristol Myers Squibb), and Christoph Sax (cynkra).

Funding was provided by Bristol Myers Squibb.