2025-09-29

ChatR: A Context-Aware R Programming Assistant

Michael Lawrence
R Core
Genentech

Gabriel Becker
R contributor

Hanxia Li
Georgia Institute of Technology

UNDER CONSTRUCTION

This repository is currently being revisited. An update will be released in early September 2025.

Executive Summary

ChatR is a proposed open-source intelligent assistant designed specifically for the R programming ecosystem. Over 12 months, we will build ChatR as both an R package and IDE add-in, with optional command-line integration and offline capability for secure or air-gapped environments. Unlike generic AI copilots, ChatR will be deeply context-aware, operating directly inside the R environment to support complex multi-step workflows in statistics, bioinformatics, and social sciences. R programmers today face increasing complexity in managing workflows that span data import, transformation, modeling, visualization, and reporting across dozens of packages. Generic copilots can generate R code, but they lack session awareness, frequently hallucinate functions, and are unusable in secure or offline environments. This limits reproducibility, reliability, and adoption in sensitive domains. ChatR addresses this gap by delivering a context-aware, tier-sensitive AI assistant that will: Access the current R session context (objects, packages, and state).

Use retrieval-augmented generation (RAG) to ground responses in official R documentation, vignettes, and authoritative references.

Invoke R tools and functions directly (e.g., run code, search CRAN, inspect environments).

Provide traceable, reproducible answers with citations, strengthening trust and transparency.

Support offline use in sensitive or restricted environments.

Align with modern LLM agent frameworks (e.g., LangChain, Roo Code, Cursor) while being fully integrated into R developer workflows.

The outcome will be a sustainable, community-adopted R package with IDE plugins and offline deployability. By enabling accurate, grounded assistance for coding, data analysis, package selection,

and error resolution, ChatR will reduce barriers for new users, save experts time, and strengthen the R ecosystem. Ultimately, this project advances reproducibility, usability, and accessibility in R, directly aligning with the R Consortium's mission to support community infrastructure and long-term growth.

Signatories

Project team

Michael Lawrence (Mentor, R Core, Genentech) – Provides guidance on R internals and package design.

Gabriel Becker (Mentor, R contributor) – Advises on architecture, evaluation, and community engagement.

Hanxia Li (Developer, Georgia Institute of Technology) – Responsible for design, implementation, and delivery of ChatR.

Contributors

Consulted

The Problem

The Problem

R users need smarter assistance to navigate complex workflows. Modern R projects often involve multiple steps: data import, transformation, modeling, visualization, and reporting, often across dozens of packages. Users must constantly refer to documentation, search for appropriate functions, debug errors, and integrate code from various sources. This process is fragmented and time-consuming. While generic AI copilots (like ChatGPT or GitHub Copilot) can generate R code, they lack context of the user's session and often hallucinate facts or functions that don't exist. They cannot directly read the user's variables or installed packages, and their suggestions may be incorrect or not tailored to the specific task. Moreover, they do not handle package version differences—critical for reproducibility, since many analyses rely on older package versions to reproduce published results.

What the problem is. There is currently no R-specific AI assistant that operates within the R environment to provide reliable, context-rich help. Existing tools either give generic answers that may be outdated or wrong, or require users to manually search through CRAN documentation, StackOverflow, and GitHub issues. Coverage is another limitation: R's ecosystem has thousands of CRAN packages, including obscure or newly updated ones, many of which are unlikely to appear in generic LLM training data. This gap makes support for specialized research workflows weak or misleading.

Who it affects. This affects a broad spectrum of R users—data scientists, statisticians, social scientists, and domain specialists in fields such as bioinformatics, epidemiology, and finance. Novice users are discouraged by opaque error messages and steep learning curves, while advanced users waste time context-switching between coding and searching. In high-stakes settings such as healthcare or government, the inability to use cloud-based copilots (due to privacy/security restrictions) further excludes many from AI support.

Why it is a problem. The consequences are inefficiency and barriers to entry. Without intelligent support, advanced R users spend significant time troubleshooting, while beginners struggle to locate the right functions or debug errors. This slows down research and development, reduces reproducibility, and ultimately hinders adoption of R. In sensitive environments, sending proprietary algorithms or data to external APIs is unacceptable, which means existing AI copilots are simply unusable.

What solving the problem will enable. If addressed, users would gain an assistant aware of their session state, capable of retrieving authoritative documentation, sensitive to package versions, and able to execute R code safely. This would enhance productivity, lower barriers for newcomers, and improve reproducibility across scientific and industry domains. An integrated solution would especially benefit restricted or offline environments by offering a local, privacy-preserving mode of operation.

Summary of existing work and previous attempts. The R community has begun exploring AI integrations, but current solutions only address parts of the problem. The {ragnar} package provides retrieval-augmented generation (RAG) workflows in R, making it possible to embed and retrieve domain-specific documentation. The {tidyprompt} package focuses on prompt engineering: it introduces "prompt wraps" that not only structure the input prompt but also add extraction and validation functions on the LLM's output, enabling features like structured responses, retries, reasoning modes, autonomous R function calling, and R code evaluation. The {ellmer} package makes it easy to use LLMs from R, supporting multiple providers and offering capabilities such as streaming output, tool/function calling, and structured data extraction. Finally, the {btw} package helps capture and describe a user's computational environment—gathering context about objects, package documentation, and the working directory—and integrates with {ellmer} as a tool provider.

These efforts demonstrate strong community interest in AI-assisted coding in R, but they remain fragmented. {ragnar}, {tidyprompt}, {ellmer}, and {btw} each contribute valuable functionality, yet they require users to manually assemble components and still do not deliver a dedicated, session-aware coding assistant. In particular, coverage gaps remain: for example, {btw} focuses on installed packages, with uninstalled ones limited to metadata lookups rather than full vignettes or examples. No existing tool yet combines authoritative CRAN knowledge, version-aware reproducibility, coverage of obscure or recently updated packages, privacy-preserving offline operation, local context inspection, and direct tool execution into a coherent workflow.

R Core support. As this proposal does not involve direct changes to R itself, no letter of support from an R Core member is required at this stage

The proposal

Overview

We propose to develop **ChatR**, a context-aware AI assistant that lives inside the R ecosystem.
ChatR addresses the gap between generic copilots and R-specific needs by combining retrieval-augmented generation (RAG) with R session awareness and tool invocation.

Users will interact through a chat interface (R console, RStudio add-in, or VS Code extension), asking questions in natural language. ChatR will classify the query, gather R context (objects, packages, errors), retrieve relevant documentation, optionally execute code, and return a grounded answer with citations.

By integrating with R's infrastructure and supporting offline use, ChatR empowers R programmers to:

- Reduce time spent searching documentation and debugging.
- Increase accuracy and reproducibility by grounding in authoritative sources.
- Improve accessibility for newcomers in statistics, bioinformatics, and social science.

Timeline

- **Months 1–2**: Design, infrastructure setup, initial knowledge base.
- **Months 3–5**: MVP delivery and early user testing.
- **Months 6–9**: Expanded tools, autonomous agent loop, evaluation.
- **Months 10–12**: Packaging, documentation, CRAN release, community outreach.

Failure modes & recovery

- *LLM unreliability*: Mitigated by constraining outputs to retrieved documentation.
- *Performance bottlenecks*: Use efficient vector stores, quantized models, caching.
- *Scope creep*: Prioritize core workflows; defer advanced features to future work.

Detail

Minimum Viable Product

The MVP (by Month 3) will demonstrate end-to-end functionality:

- Command-line chat interface in R.
- Basic query classification (docs/help, code generation, error fix).
- Local knowledge base of base R + tidyverse docs indexed with embeddings.
- Simple tool calls (CRAN package search, safe code execution sandbox).
- Responses fact-checked against retrieved text and cited for transparency.

This validates feasibility and provides early feedback.

Architecture

ChatR will follow a modular design:

- **Frontend**: R console loop, later RStudio add-in or Shiny gadget.
- **Core Orchestrator**: Classifies query, manages flow (ReAct-style).

- **Retrieval Module**: Embedding-based search (FAISS/R alternatives, `{ragnar}`).
- **Tool Interfaces**: Documentation search, CRAN search, code execution, environment inspection.
- **LLM Integration**: Start with GPT-4 API, evaluate open-source models for offline use.
- **Dialogue Management**: Maintain conversation memory for multi-turn queries.
- **MCP/LangChain compatibility**: Ensure extensibility and cross-ecosystem integration.

Assumptions

- R documentation and package manuals provide sufficient knowledge.
- Current LLMs (API or open models) are capable of producing useful R answers when grounded.
- Users can install dependencies and, for offline mode, run a small local model.
- Community will engage with feedback and contributions.
- User environment: Users can install ChatR and its dependencies; for offline mode, machines should handle smaller local LLMs (7B–13B) and embedding indexes. Otherwise, an online API mode will be available.
- Tool safety and trust: Code execution is restricted to the user's R session, with safeguards and confirmation prompts to prevent unsafe actions.

If any assumptions fail (e.g., offline models prove too resource-heavy), fallback options such as smaller KBs or API-based models will be provided.

External dependencies

R packages

- `{ragnar}` for RAG workflows (chunking, embeddings, retrieval).
- `{duckdb}` for efficient on-disk storage of embeddings.
- `{pkgsearch}`, `{tools}`, `{BiocPkgTools}` for CRAN/Bioconductor metadata.
- `{callr}`, `{processx}`, `{evaluate}` for safe code execution and output capture.
- `{rstudioapi}`, `{shiny}`, `{miniUI}` (optional) for IDE add-ins.

Vector search

- FAISS (via Python/`reticulate`) as primary ANN backend.
- Fallbacks: `RcppHNSW` or `RcppAnnoy` for native approximate nearest neighbor search.

LLM backends

- Online: OpenAI GPT-4/5 APIs for prototyping.
- Offline: Local OSS models (e.g., LLaMA, Code Llama) via `reticulate` or `{torch}`.

Embeddings

- Online: OpenAI embedding API.
- Offline: `sentence-transformers` (via `reticulate`) or `{text2vec}` for lightweight baselines.

Integration (optional)

- Model Context Protocol (MCP) or LangChain for interoperability with external orchestrators.
- `{plumber}` for exposing a local HTTP interface if editor integration requires it.

Project plan

Start-up phase

In the initial phase, we will focus on quickly establishing the foundation for ChatR:

- **Collaboration & Infrastructure**: Set up a public GitHub repository following R Consortium guidelines, implement continuous integration (CI), and establish a test suite. Contributors will have clear guidelines for code style, contribution processes, and issue tracking.
- **License Decisions**: The project will adopt a permissive open-source license (MIT or GPL, depending on dependency compatibility) to maximize adoption while respecting CRAN and upstream package licensing.
- **Reporting Framework**: A transparent reporting system will be set up using GitHub milestones, quarterly ISC updates, and progress documentation.
- **Design & Prototyping**: Define the architecture for ChatR, run small technology spikes (e.g., embedding a subset of R documentation, testing retrieval queries), and prepare an MVP specification.
- **Community Engagement**: Announce project initiation on R Consortium channels (mailing lists, forums), inviting feedback and potential contributors early.

By the end of this phase, we expect to have (i) a working prototype interface (`chat_r()`) in the R console, (ii) an initial knowledge base of base R + major packages indexed, and (iii) a public roadmap with clear milestones.

Technical delivery

The technical work will follow iterative, milestone-driven development:

- **Milestone 1 – MVP Completion (Month 3)**
  Deliver a minimal interactive assistant with query classification, retrieval from base R docs, and basic answer generation. Demonstrate source citations and context retention for simple follow-up queries.

- **Milestone 2 – Enhanced RAG & Toolset (Months 4–5)**
  Expand the knowledge base to ~100 packages, implement CRAN package search, safe R code execution, and environment introspection tools. Demonstrate retrieval + execution workflows (e.g., plotting a user's data).

- **Milestone 3 – Agent Autonomy & Reasoning (Months 6–7)**
  Implement a ReAct-style orchestration pipeline so ChatR can reason about queries, invoke tools in sequence, and refine responses. Add an error-diagnosis module to address common R errors.

- **Milestone 4 – Evaluation & Tuning (Month 8)**
  Build an evaluation harness using both automated LLM-as-a-judge scoring and user feedback. Optimize retrieval latency and refine source citation presentation.

- **Milestone 5 – Integration & Deployment (Months 9–10)**
  Package ChatR for CRAN submission and provide MCP/LangChain integration hooks. Develop an IDE add-in (e.g., RStudio gadget) for interactive use.

- **Milestone 6 – Documentation & Release (Months 11–12)**
  Deliver user and developer documentation, blog posts, and tutorials. Finalize v1.0 release and announce availability via R Consortium blog, RStudio community, and UseR! channels.

Progress will be reviewed bi-weekly with mentors and quarterly with ISC.

Other aspects

- **Accessibility & Licensing**: All code will be open source (MIT or GPL). Embedding indexes, prompts, and evaluation harnesses will also be released to enable reproducibility.

- **Hosting & Contribution**: ChatR will be hosted on GitHub with open issues and milestone tracking. CRAN distribution will ensure wide accessibility.

- **Publicity & Community Engagement**:
  - **Announcement post**: Project launch announcement on R Consortium channels.
  - **Progress posts**: Quarterly blog posts summarizing milestones, challenges, and demos.
  - **Social media**: Updates shared via R-Ladies, RStudio Community, and Mastodon/Twitter.

  - **UseR! Conference**: Present progress at UseR! or similar events.
  - **ISC Meetings**: Provide quarterly updates and final demo to ISC.

This open and visible process ensures transparency, reproducibility, and strong community buy-in.

Budget & funding plan

The total requested budget (~$10,000) will be allocated primarily to labor, with milestone-based disbursement:

- **Developer Compensation (~90%)**
  - Supports ~20 hours/week effort for 12 months by the principal developer (Hanxia Li).
  - Funds disbursed in two tranches: 50% at contract signing, 50% at project completion.
- **Community Engagement & Miscellaneous (~10%)**
  - Small allocation for tutorial creation (screencasts, design), community testing support (swag/gifts), and incidental costs (e.g., domain name for demo site).
  - Any unspent funds will remain unused.

Disallowed costs (travel, lodging, indirect costs, hardware) are excluded per ISC guidelines. Cloud compute will not be funded unless project-specific and pre-approved by ISC; otherwise, development will rely on existing institutional resources or free credits.

Milestone-based funding plan:

Milestone	Work to be Done	Expected Outcome	Budget Allocation
MVP Completion (Month 3)	Console prototype with retrieval & citation	Demo to ISC, early feedback	15%
Enhanced RAG & Tools (Months 4–5)	Expanded KB + tool integrations	Advanced demo	20%
Agent Autonomy (Months 6–7)	Multi-step reasoning, error handling	Usable assistant	20%
Evaluation & Tuning (Month 8)	Evaluation harness + optimizations	Quality assurance	15%
Integration & Deployment (Months 9–10)	CRAN packaging, IDE add-in	Public release candidate	15%
Documentation & Release (Months 11–12)	Tutorials, final blog posts, v1.0 release	Final report & CRAN submission	15%

This plan ensures that funds are directly tied to measurable progress and successful outcomes.

Success

Definition of done

We will consider the project successfully completed when the following are delivered:

1. **ChatR Package v1.0 Released**
   A fully functional R package available on CRAN or R-universe that includes:
   - Context-aware chat assistant functionality
   - Support for documentation/how-to questions, error debugging, and RAG-grounded responses
   - Both offline mode (local knowledge base and local/optional models) and online mode (API keys).
2. **Integrated Agent Ecosystem Support**
   Demonstrated interoperability with IDEs and agent frameworks. For example, ChatR running inside VS Code or via the Model Context Protocol (MCP) and LangChain pipelines.
3. **Error Diagnosis Module**
   Robust error handling where ChatR explains or suggests fixes for at least 80% of a defined test suite of common R errors (e.g., "object not found," "factor level not present," file I/O errors).
4. **Knowledge Base Coverage**
   Indexed coverage of:
   - Base R and recommended packages
   - At least 100 popular CRAN packages (e.g., ggplot2, dplyr, Shiny, data.table, lme4)
   - Selected Bioconductor documentation.
     A documented process for updating/expanding this knowledge base will also be provided.
5. **Documentation & Examples**
   Complete user-facing vignettes and developer docs, plus at least one blog post announcing ChatR with walkthroughs of use cases.
6. **Community Engagement**
   Tangible evidence of adoption and interest (e.g., GitHub stars/forks, issues or PRs from external users, mentions in R community forums).

Measuring success

We will evaluate success using multiple methods:

- **Automated Evaluation Metrics**
  An LLM-judge framework will compare ChatR's answers on 50–100 R questions against baseline LLM responses. Target: 80% correctness and 20% improvement over baseline (e.g., ChatGPT without context).

- **User Testing & Surveys**
  Volunteer R users will test ChatR and complete surveys rating usefulness and accuracy (1–5 scale). Success target: mean scores 4 for both metrics.
- **Adoption Indicators**
  Track GitHub stars, package downloads, and forum activity. A milestone is 50 users testing ChatR and at least one external pull request or extension idea.
- **Qualitative Feedback & Case Studies**
  Documented "success stories" where ChatR demonstrably reduces time-to-solution (e.g., cutting a 30-minute workflow down to 5 minutes).
- **Beginner Impact**
  Evidence that novice users successfully complete tasks they otherwise could not, highlighting ChatR's role in democratizing R access.

Future work

This 12-month grant lays the foundation, but ChatR is designed to evolve. Potential extensions include:

- **Long-term Maintenance**
  Establish a community-driven maintenance team under R Consortium guidance. Future grants or sponsorships may support continued development.
- **Improved AI Capabilities**
  Fine-tuning open models on R-specific Q&A data for stronger offline performance and exploring model compression for lightweight deployment.
- **Cross-Ecosystem Expansion**
  Extending the architecture to other languages (e.g., Python, Julia) or hybrid environments (RMarkdown with R/Python).
- **Deeper RStudio/Posit Integration**
  Working with Posit to embed ChatR in RStudio's Help pane or as an official add-in.
- **Community Knowledge Base**
  Mechanisms for user-contributed Q&A to expand ChatR's intelligence over time.
- **Enhanced Features**
  Multi-modal support (plots/images), "learning mode" for didactic explanations, stronger privacy controls, and accessibility enhancements like voice input.
- **Impact Studies**
  Partnering with educators to measure ChatR's effect on R learning and productivity in academic or professional settings.

By combining measurable outcomes with clear pathways for growth, this project ensures immediate community value and a sustainable roadmap that maximizes the impact of R Consortium's investment.