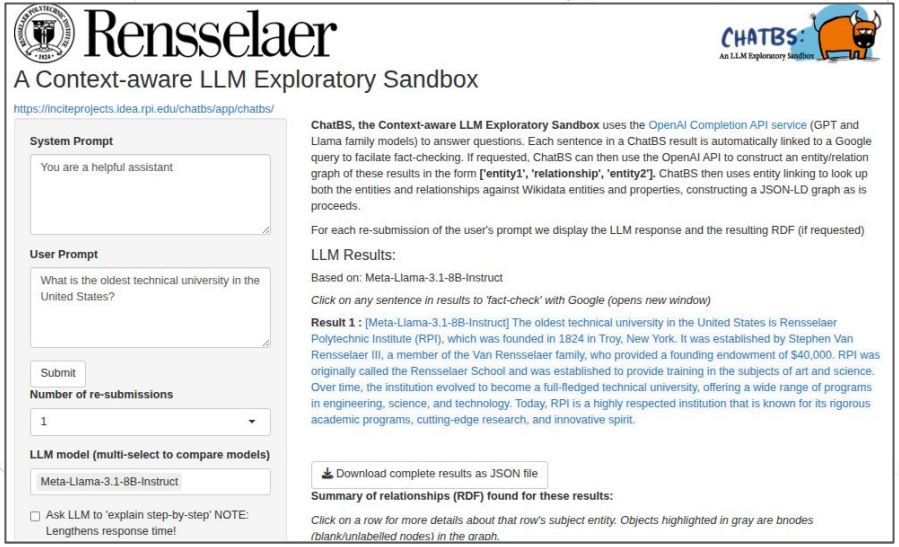

RPI
FUTURE OF
COMPUTING
INSTITUTE
LLMs and R: How Posit is Making it Easy!
Wednesday, 12 Nov 2025
RPIrates: The RPI R Users Group
The Rensselaer Future of Computing Institute
Rensselaer Polytechnic Institute
RPIrates
The RPI R Users Group
Tall

Recalling a Simpler Time...

RPIrates Feb 2023

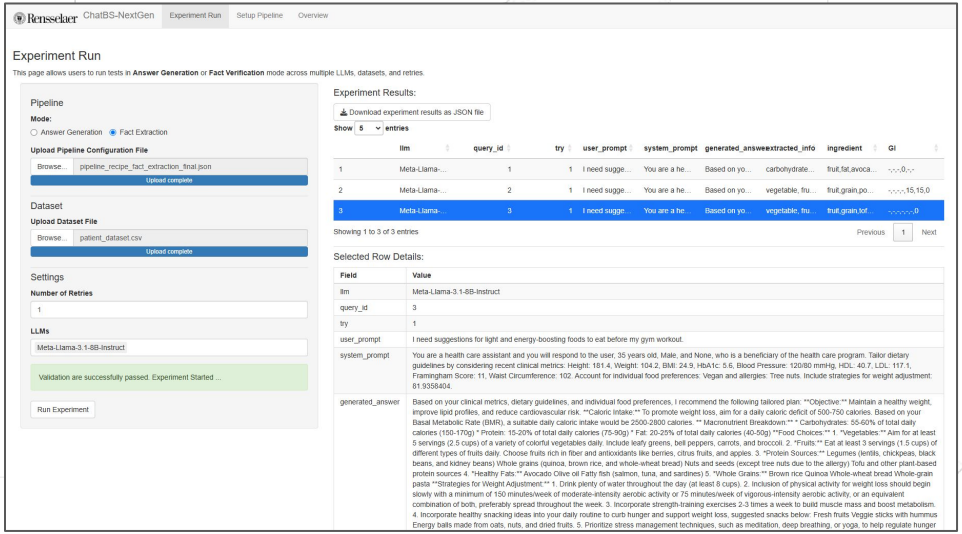
ChatBS and ChatBS-NextGen

ChatBS: A Context-aware LLM Exploratory Sandbox



2024

ChatBS-NexGen: Automating Workflows for LLM Evaluation over Knowledge Graphs



2025

Posit Generative AI Solutions

Posit provides a range of tools and LLM-related packages for R and Python

Access LLMs

Build Shiny apps
that chat with LLMs

Evaluate LLMs

Posit Generative AI Solutions

Posit provides a range of tools and LLM-related packages for R and Python

Enhance LLM workflows

IDE Integrations

Automate and improve data science tasks with GenAI

ellmer

ellmer makes it easy to use large language models (LLM) from R. It supports a wide variety of LLM providers and implements a rich set of features including streaming outputs, tool/function calling, structured data extraction, and more.

ellmer is one of a number of LLM-related packages created by Posit:

- Looking for something similar in python? Check out chatlas!
- Want to evaluate your LLMs? Try vitals.
- Need RAG? Take a look at ragnar.
- Want to make a beautiful LLM powered chatbot? Consider shinychat.
- Working with MCP? Check out mcptools.

ellmer

Providers

ellmer supports a wide variety of model providers:

- Anthropic's Claude: `chat_anthropic()`.
- AWS Bedrock: `chat_aws_bedrock()`.
- Azure OpenAI: `chat_azure_openai()`.
- Cloudflare: `chat_cloudflare()`.
- Databricks: `chat_databricks()`.
- DeepSeek: `chat_deepseek()`.
- GitHub model marketplace: `chat_github()`.
- Google Gemini/Vertex AI: `chat_google_gemini()`, `chat_google_vertex()`.
- Groq: `chat_groq()`.
- Hugging Face: `chat_huggingface()`.
- Mistral: `chat_mistral()`.
- Ollama: `chat_ollama()`.
- OpenAI: `chat_openai()`.
- OpenRouter: `chat_openrouter()`.
- perplexity.ai: `chat_perplexity()`.
- Snowflake Cortex: `chat_snowflake()` and `chat_cortex_analyst()`.
- VLLM: `chat_vllm()`.

ellmer

Provider/model choice

If you're using ellmer inside an organisation, you may have internal policies that limit you to models from big cloud providers, e.g. `chat_azure_openai()`, `chat_aws_bedrock()`, `chat_databricks()`, or `chat_snowflake()`.

If you're using ellmer for your own exploration, you'll have a lot more freedom, so we have a few recommendations to help you get started:

- `chat_openai()` or `chat_anthropic()` are good places to start. `chat_openai()` defaults to **GPT-4.1**, but you can use `model = "gpt-4-1-nano"` for a cheaper, faster model, or `model = "o3"` for more complex reasoning. `chat_anthropic()` is also good; it defaults to **Claude 4.0 Sonnet**, which we have found to be particularly good at writing R code.

- `chat_google_gemini()` is a strong model with generous free tier (with the downside that your data is used to improve the model), making it a great place to start if you don't want to spend any money.

- `chat_ollama()`, which uses Ollama, allows you to run models on your own computer. While the biggest models you can run locally aren't as good as the state of the art hosted models, they don't share your data and are effectively free.

vitals

vitals is a framework for large language model evaluation in R. It's specifically aimed at ellmer users who want to measure the effectiveness of their LLM products like custom chat apps and querychat apps. You can use it to:

- Measure whether changes in your prompts or additions of new tools improve performance in your LLM product
- Compare how different models affect performance, cost, and/or latency of your LLM product
- Surface problematic behaviors in your LLM product

The package is an R port of the widely adopted Python framework Inspect. While the package doesn't integrate with Inspect directly, it allows users to interface with the Inspect log viewer and provides an on-ramp to transition to Inspect if need be by writing evaluation logs to the same file format.

ragnar

`ragnar` is an R package that helps implement Retrieval-Augmented Generation (RAG) workflows. It focuses on providing a complete solution with sensible defaults, while still giving the knowledgeable user precise control over each step. We don't believe that you can fully automate the creation of a good RAG system, so it's important that `ragnar` is not a black box. `ragnar` is designed to be transparent. You can easily inspect outputs at intermediate steps to understand what's happening.

ragnar

Key Steps

1. Document Processing

`ragnar` works with a wide variety of document types, using MarkItDown to convert content to Markdown.

Key functions:

- `read_as_markdown()` : Convert a file or URL to markdown
- `ragnar_find_links()` : Find all links in a webpage

2. Text Chunking

Next we divide each document into chunks. Ragnar defaults to a strategy that preserves some of the semantics of the document, but provides plenty of opportunities to tweak the approach.

Key functions:

- `markdown_chunk()` : Full-featured chunker that identifies semantic boundaries and intelligently chunks text.

ragnar

3. Context Augmentation (Optional)

RAG applications benefit from augmenting text chunks with additional context, such as document headings and subheadings. `ragnar` makes it easy to keep track of headings and subheadings as part of chunking.

`markdown_chunk()` automatically associates each chunk with the headings that are in scope for that chunk.

4. Embedding

`ragnar` can help compute embeddings for each chunk. The goal is for `ragnar` to provide access to embeddings from popular LLM providers.

Key functions:

- `embed_ollama()`
- `embed_openai()`
- `embed_bedrock()`
- `embed_databricks()`
- `embed_google_vertex()`

Note that calling the embedding function directly is typically not necessary. Instead, the embedding function is specified when a store is first created, and then automatically called when needed by `ragnar_retrieve()` and `ragnar_store_insert()`.

ragnar

5. Storage

Processed data is stored in a format optimized for efficient searching, using `duckdb` by default. The API is designed to be extensible, allowing additional packages to implement support for different storage providers.

Key functions:

- `ragnar_store_create()`
- `ragnar_store_connect()`
- `ragnar_store_insert()`

6. Retrieval

Given a prompt, retrieve related chunks based on embedding distance or bm25 text search.

Key functions:

- `ragnar_retrieve()` : high-level function that performs both `vss` and `bm25` search and de-overlaps retrieved results.
- `ragnar_retrieve_vss()` : Retrieve using `vss` DuckDB extension
- `ragnar_retrieve_bm25()` : Retrieve using `full-text search DuckDB extension`
- `chunks_deoverlap()` : Consolidates retrieved chunks that overlap.

7. Chat Augmentation

`ragnar` can equip an `ellmer::Chat` object with a retrieve tool that enables an LLM to retrieve content from a store on-demand.

https://ragnar.tidyverse.org/

RPIrates The RPI R Users Group

RPI FUTURE OF COMPUTING INSTITUTE

Also RAGNAR

- The Relay Races: https://runragnar.com/

- The Legend

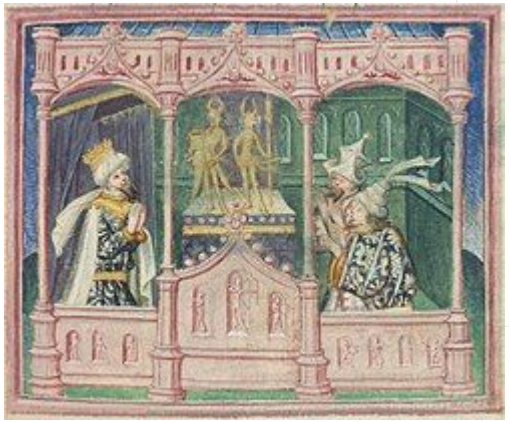
Ragnar Lodbrok

Article Talk

From Wikipedia, the free encyclopedia

Ragnar Lodbrok (Old Norse: *Ragnarr loðbrók*, lit. 'Ragnar hairy-breeches'),[a] according to legends,[2] was a Viking hero and a Swedish and Danish king.[3]

He is known from Old Norse poetry of the Viking Age, Icelandic sagas, and near-contemporary chronicles. According to traditional literature, Ragnar distinguished himself by conducting many raids against the British Isles and the Carolingian Empire during the 9th century. He also appears in Norse legends, and according to the legendary sagas *Tale of Ragnar's Sons* and a *Saga about Certain Ancient Kings*,[4] Ragnar Lodbrok's father has been given as the legendary king of the Swedes, Sigurd Ring.[5][6]



https://ragnar.tidyverse.org/

RPIrates The RPI R Users Group

RPI FUTURE OF COMPUTING INSTITUTE

shinychat

shinychat provides a Shiny toolkit for building generative AI applications like chatbots and streaming content. It's designed to work alongside the ellmer package, which handles response generation.

Installation

You can install shinychat from CRAN with:

```
install.packages("shinychat")
```

Or, install the development version of shinychat from GitHub with:

IDE Integrations: chattr

chattr

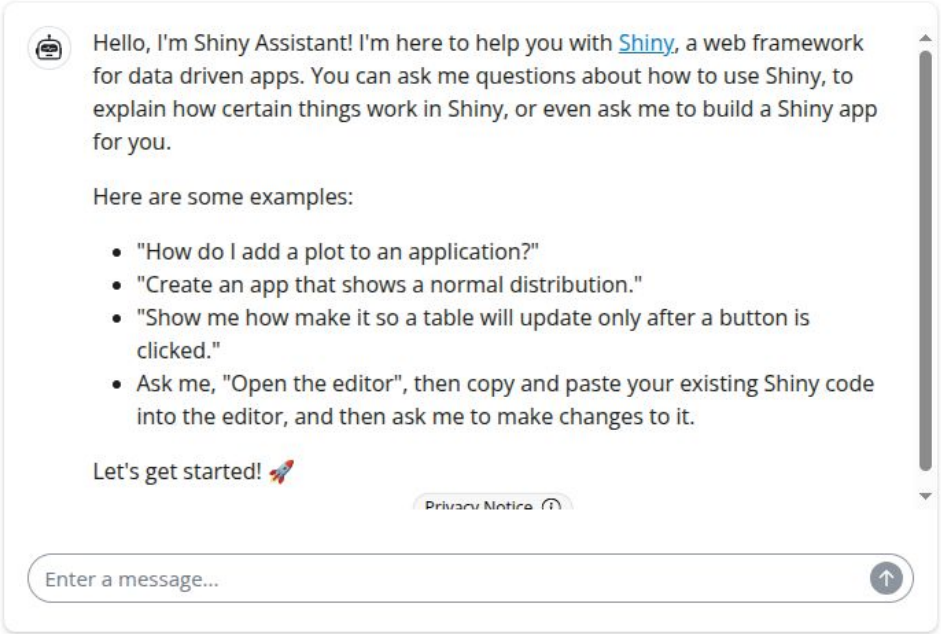
Intro

`chattr` is an interface to LLMs (Large Language Models). It enables interaction with the model directly from RStudio and Positron. `chattr` allows you to submit a prompt to the LLM from your script, or by using the provided Shiny Gadget.

This package's main goal is to aid in exploratory data analysis (EDA) tasks. The additional information appended to your request, provides a sort of "guard rails", so that the packages and techniques we usually recommend as best practice, are used in the model's responses.

https://mlverse.github.io/chattr/

IDE Integrations: Shiny Assistant



Access LLMs in R and Python

ellmer

The ellmer package simplifies LLM integration in R. Ellmer provides a consistent interface to multiple LLM providers, with features like streaming, tool calling, and structured data extraction.

- Documentation
- Release blog post

chatlas

The chatlas package simplifies LLM integration in Python. Chatlas offers a unified interface across LLM providers for tasks like streaming chats, tool calling, and structured output.

- Documentation
- Release blog post

ragnar

The ragnar package enhances LLM performance with Retrieval-Augmented Generation (RAG) in R, which allows you to retrieve relevant external data to inform responses.

- Documentation
- Release blog post

Enhance your LLM workflow

btw

The btw package helps you describe your computational environment to LLMs.

- Documentation
- Technical introduction

vitals

The vitals package provides a framework for large language model evaluation in R.

- Documentation
- Technical introduction

mcptools

The mcptools package implements a Model Context Protocol (MCP) server for your R sessions.

- Documentation
- Technical introduction

https://ivelasq-posit-ai-tools.share.connect.posit.cloud/

Build Shiny apps that chat with LLMs

Shinychat

Shinychat provides components integrate interactive chat interfaces into Shiny applications with a dedicated UI component.

- R documentation
- Python documentation

querychat

querychat is a drop-in component for Shiny that allows users to query a data frame using natural language.

- Documentation

Automate and improve data science tasks with GenAI

chores

The chores package automates repetitive coding tasks with LLM-powered assistants. Use customizable shortcuts to trigger code rewrites and enhancements directly in your R environment.

- Documentation
- Technical introduction

gander

The gander package enhances data science workflows in RStudio and Positron with intelligent, in-line LLM chats. Gander integrates Ellmer chats into your projects, providing context-aware responses and streaming results directly into your documents.

- Documentation
- Technical introduction

mall

The mall package applies LLM predictions to data frames efficiently. Process rows with pre-defined prompts for batch analysis in both R and Python.

- Documentation
- Technical introduction

lang

The lang package translates R function help documentation on-the-fly to other languages such as Spanish or French. Lang overrides help functions to provide instant translations within RStudio and Positron.

- Documentation

IDE Integrations

Positron Assistant

Experimental feature in Positron 2025.06.0

Positron Assistant transforms Positron into an AI-powered data science workspace, deeply integrated with your tools and workflows while leveraging enterprise-grade LLM providers.

- User Guide
- Announcement post

Databot

Databot is an interactive AI agent designed specifically for exploratory data analysis. Databot actively drives your data exploration through a tight WEAR (write, execute, analyze, regroup) loop.

- Announcement post

Shiny Assistant

Shiny Assistant accelerates Shiny development with an AI assistant. Get help with Shiny questions, create applications from scratch, or modify existing code in R and Python.

- Shiny Assistant
- Release blog post

GitHub Copilot

GitHub Copilot enhances coding productivity with AI-powered code suggestions directly within RStudio.

- Documentation
- Release blog post

chattr

The chattr package allows you to interact with LLMs directly from RStudio. Submit prompts from scripts or use the Shiny gadget for interactive conversations.

- Documentation
- Technical introduction

Enable effective observability

otel

Use the otel package as a dependency if you want to instrument your R package or project for OpenTelemetry.

otelsdk

Use the otelsdk package to produce OpenTelemetry output from an R package or project that was instrumented with the otel package.

AI demonstrations, use cases, and best practices

- Harnessing LLMs for Data Analysis | Led by Joe Cheng, CTO at Posit
- Easy tool calls with ellmer and chatlas
- Natural language data science with RStudio and Databricks
- How to use natural language data science with RStudio and Amazon SageMaker
- Announcing secure AI-Assisted data science in R with Posit and Snowflake Cortex
- AI tool built with Posit decreases unexpected deaths of hospitalized patients by 26%
- Generate data with an LLM and ellmer
- Text Summarization, Translation, and Classification using LLMs
- The potential for AI-powered Shiny app prototyping with Shiny Assistant
- Running AI/LLM Hackathons at Posit: What We've Learned
- Trail Running Meets Data Science: Adventures with LLMs and Race Stats
- Setting up local LLMs for R and Python
- What LLMs Actually Do (and What They Don't)