

Creating a Custom AI Bot for Your Class

CUNY IT Conference

Jason Bryer, Ph.D.

December 5, 2025

Introduction

My motivating question for this project is: **How are students using chatbots?**

To answer this question I created a custom chatbot so...

- Student questions and answers are saved.
- The LLM prioritizes getting from a curated list of resources.
- Data is not sent to any third party (i.e. using local models).

What is AI?

First, AI is a marketing term. I prefer to be more specific regarding what we are doing:

Machine Learning (ML)

This involves training models (e.g. regression, etc.) to predict outcomes. We have been doing this for over a century.

Large Language Models (LLM)

This is often what people mean when they say AI. This includes products like ChatGPT, Anthropic, and Google Gemini. LLMs generate text, images, videos, etc. from a prompt.

Outline

1. Document preprocessing - convert our resources (websites, PDFs) to markdown so the LLM can process the text.
2. Text chunking - divide the text into smaller chunks.
3. Embeddings - convert the text into embeddings. These are numeric representations of the text.
4. Storage - Insert the embeddings into a database. We will use **DuckDB** which is a high performance database format designed for data analysis.
5. Connect/Retrieve - connect the knowledge store to our LLM to assist in answer queries.

Application Overview

This application is built using R, however minimal knowledge of R is required to setup your application. You can install R [here](#) and RStudio [here](#). I have more detailed instructions on [my course website](#)

- R - Main programming language.
- Shiny - Framework for creating a web application in R.
- ellmer - This package provides functions to interact with many LLM providers, including Ollama.
- ragnar - This package assists with the creation of a RAG database.
- rollama - This package is designed to work specifically with Ollama.
- login - R package that adds user authentication to Shiny applications. Recording of a talk I gave on this package at ShinyConf 2024 is available here: <https://bryer.org/posts/2024-04-17-ShinyConf2024.html>

Local Setup

You first need to install R. I also recommend you install RStudio. The following command will install the `chatR` package and all dependencies.

```
install.packages('pak')
pak::pkg_install('jbryer/chatR')
```

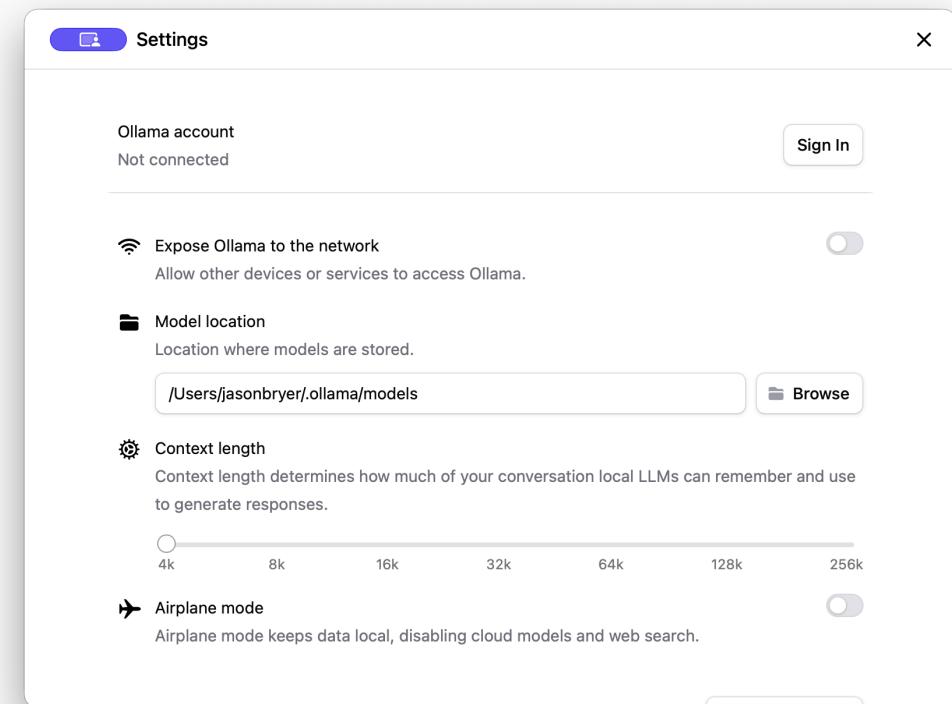
Ollama

We are going to use Ollama as our LLM engine. We can run this model locally which means student questions are not sent to any third party sites.

Download from <https://ollama.com>

List of available models:

<https://ollama.com/library>



Verifying R can talk to Ollama

First, let's verify that the local Ollama service is running.

```
rollama::ping_ollama() # Make sure Ollam is running
```

```
model <- 'llama3.1'
```

```
rollama::pull_model(model = model)
model_info <- rollama::show_model(model = model)
```

Connecting to the LLM

The `ellmer` package has a number of functions for connecting to *all* the popular LLMs.

```
chat <- ellmer::chat_ollama(  
  model = model,  
  echo = 'all'  
)
```

Once we have a `chat` object we can begin sending queries.

```
repsonse <- chat$chat('What is the central limit theorem? Keep your answers to less than three sentences.',  
                      echo = 'none')  
repsonse
```

```
## The Central Limit Theorem (CLT) states that, given certain conditions, the  
## distribution of sample means will approximate a normal distribution even if the  
## underlying population distribution is not normal. This allows researchers to  
## use z-scores and standard statistical tools to analyze large datasets that may  
## have come from non-normal populations.
```

Lots of LLMs

This is the current list of available LLMs within the `ellmer` package.

```
ls('package:ellmer')[grep('^chat_', ls('package:ellmer'))]
```

```
## [1] "chat_anthropic"          "chat_aws_bedrock"        "chat_azure_openai"  
## [4] "chat_claude"             "chat_cloudflare"         "chat_databricks"  
## [7] "chat_deepseek"            "chat_github"              "chat_google_gemini"  
## [10] "chat_google_vertex"       "chat_groq"                "chat_huggingface"  
## [13] "chat_mistral"             "chat_ollama"              "chat_openai"  
## [16] "chat_openai_compatible"   "chat_openrouter"          "chat_perplexity"  
## [19] "chat_portkey"             "chat_snowflake"           "chat_vllm"
```

Retriever Augmented Generated Models

"Retrieval-augmented generation (RAG) is a technique that enables large language models (LLMs) to retrieve and incorporate new information" ([Wikipedia](#)).

RAG is a way of restricting chatbots to a curated set of resources.

You can optionally allow the chatbot to go beyond the resources if it cannot find an answer in them.

Document preprocessing and chunking

We need to first convert our resources into a format the LLM can understand.

```
website_urls <- c('https://r4ds.hadley.nz') # Only using one site for illustrative purposes
pages <- sapply(website_urls, ragnar::ragnar_find_links)
pages <- unlist(pages) |> unname()
```

For all the pages found, convert to markdown and then chunk the text.

```
page_chunks <- list()
for(i in 1:length(pages)) {
  page_chunks[[i]] <- pages[i] |> read_as_markdown() |> markdown_chunk()
}
```

Create embeddings and storage

Create the knowledge store (DuckDB database)

```
store <- ragnar_store_create(  
  'data-raw/ragner_ollama.duckdb',  
  embed = \((x) ragnar::embed_ollama(x, model = 'llama3.1'))  
)
```

Now we can insert our text embeddings into the database.

```
for(i in 1:length(page_chunks)) {  
  ragnar_store_insert(store, page_chunks[[i]])  
}
```

Finally, build an index for our knowledge store.

```
ragnar::ragnar_store_build_index(store)
```

Connecting to a built knowledge store

```
store <- ragnar::ragnar_store_connect(  
  '.../../data-raw/ragner_ollama.duckdb',  
  read_only = FALSE  
)
```

Inspecting the knowledge store

WARNING: Creating the knowledge store will take a long time. I include 14 sources (3 PDFs, 11 websites) and it took over 12 hours on a M1 MacBook Pro to generate the knowledge store. You can download my knowledge store from the [Github repo](#).

We can retrieve content from the store.

```
ragnar::ragnar_retrieve(store, 'bar plots in ggplot2')
```

```
## # A tibble: 5 × 9
##   origin      doc_id chunk_id  start    end cosine_distance bm25 context text
##   <chr>       <int>  <list>     <int>  <int> <list>          <dbl> <chr>  <chr>
## 1 https://boo...    335 <int>      567    2614 <dbl> [1]>      <dbl> "# [R ... "##...
## 2 https://cra...    411 <int>    236774  238580 <dbl> [1]>      <dbl> "# An ... `de...
## 3 https://ggp...    134 <int>      4128    6266 <dbl> [2]>      <dbl> "# 8nbsp; ... "``...
## 4 https://ggp...    134 <int>      8780   10506 <dbl> [1]>      <dbl> "# 8nbsp; ... "![]...
## 5 https://mod...    239 <int>      50223   52069 <dbl> [1]>      <dbl> "# Cha... "The...
```

The `ragnar` package includes a Shiny application to explore the knowledge store interactively.

System Prompt

One of the ways we can improve LLMs for our classes is by specifying a system prompt. This is global direction applied to all queries. Here is the system prompt I am currently using for my R and Statistics chatbot:

You are an expert **in** R and statistics. You are concise.

Always perform a search of the knowledge store **for** each user request.

If the initial search does not **return** relevant documents, you may perform an additional search. Each search will **return** unique, new excerpts.

If no relevant results are found, inform the user and do not attempt to answer the question.

If the user request is ambiguous, perform at least one search first, then ask a clarifying question.

Every response must cite links to official documentation sources.

Connecting the knowledge store to the LLM

Using the `ragnar_register_tool_retrieve` we can connect our knowledge store with any `ellmer_chat` object.

```
ragnar::ragnar_register_tool_retrieve(chat, store)
```

```
chat <- ellmer::chat_ollama(  
  model = model, echo = 'all',  
  system_prompt = system_prompt)
```

```
chat$chat('How do I subset columns from a data frame?')
```

```
## > How do I subset columns from a data frame?  
## < You can use the following functions in R to subset columns from a data frame:  
## <  
## < 1. **$**  
## <  
## <     * This operator allows you to select one column by name.  
## <  
## <           ` ```r  
## < df$name_col  
## < ````
```

DEMO

Demo Locally

You can run the application locally.

```
if(!dir.exists('data-raw/')) { dir.create('data-raw/' , recursive = TRUE) }
piggyback::pb_download(
  file = "ragner_ollama.duckdb",
  dest = 'data-raw/' ,
  repo = "jbryer/chatR",
  tag = "v1.0.0")
chatR::run_chatR(
  store_location = 'data-raw/ragner_ollama.duckdb' ,
  system_prompt = system_prompt
)
```

Next Steps?

Just finished my first semester using this custom chatbot. Over the break I will begin analyzing in more detail the questions students ask.

I want to also evaluate the accuracy of the answers students received from their questions.

For the Spring semester I intend to solicit more explicit feedback from students on the quality of the chatbot.

I am also interested in how this may relate to students responses to one-minute-papers they complete after each class. Students respond to the following two questions:

1. What was the most important thing you learned during this class?
2. What important question remains unanswered for you?

Questions?

Thank You!

✉️ jason.bryer@cuny.edu



@jbryer

Ⓜ️ @jbryer@vis.social

🌀 www.bryer.org

