

# A no bullshit guide to LLMs

In R, with ellmer

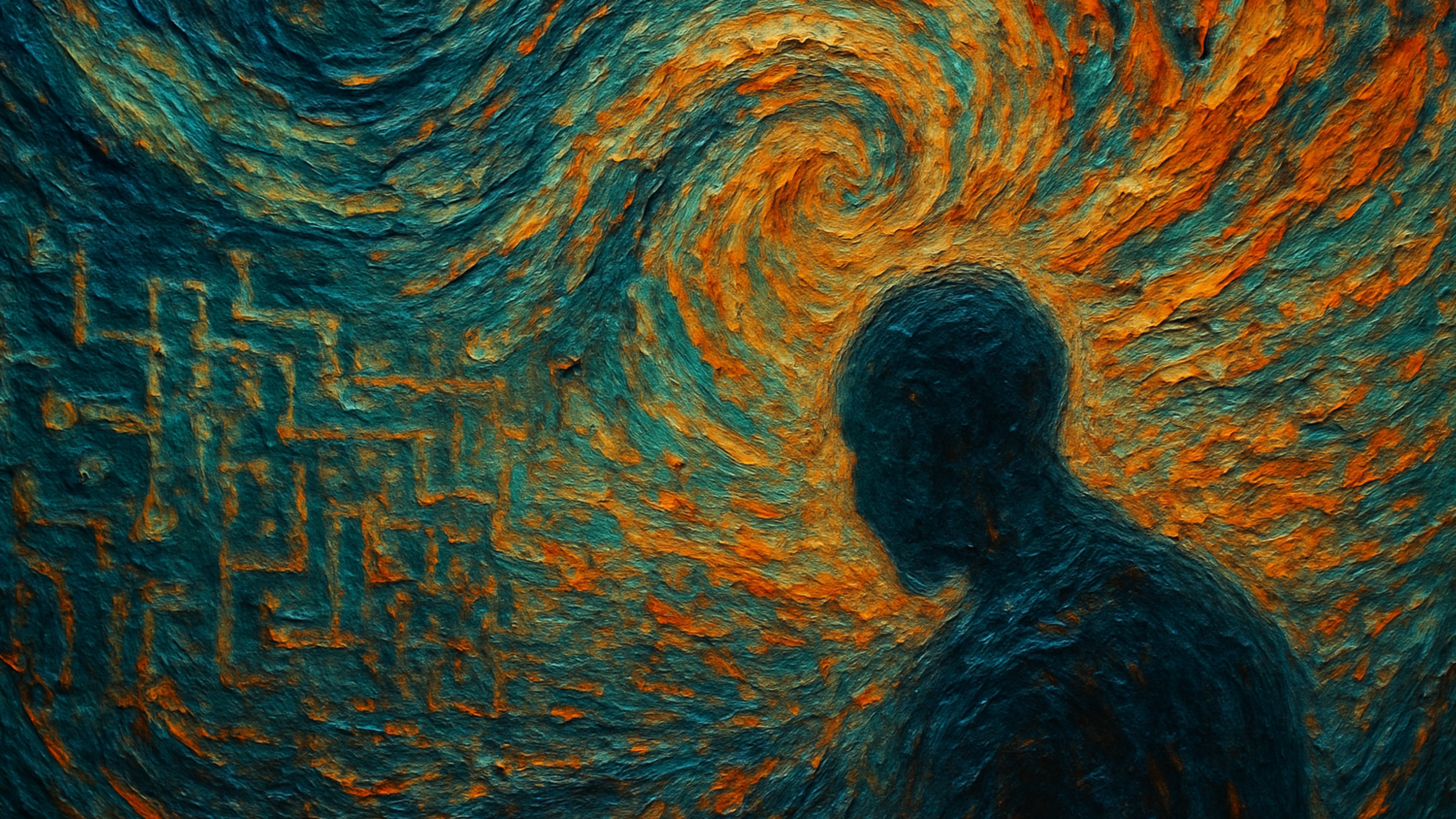
Hadley Wickham

Chief Scientist, Posit

April 2025













Two main goals of this talk:

Useful skills

Demystify jargon



# And I'm going to do this (mostly) in R

```
library(ellmer)
```

```
chat ← chat_anthropic()  
chat$chat("Who are you?")
```

My favourite!

```
#> Using model = "claude-sonnet-4-20250514".
```

```
#> I'm Claude, an AI assistant created by Anthropic. I'm here to help
```

```
#> with a wide variety of tasks like answering questions, helping with
```

```
#> analysis and research, creative writing, math and coding problems,
```

```
#> and having conversations. Is there something specific I can help
```

```
#> you with today?
```









# LLMs can't multiply big numbers

```
chat ← chat_anthropic()
x ← 20598162
y ← 83106206
chat$chat(interpolate("What's {{x}} * {{y}}?"))
#> I'll calculate 20598162 * 83106206 for you.
#>
#> 20598162 × 83106206 = 1,711,398,278,221,972
#>
#> So the answer is 1,711,398,278,221,972.
format(x * y, big.mark = ",", scientific = FALSE)
#> [1] "1,711,835,094,393,372"
```



# LLMs can't even count

```
chat ← chat_anthropic(model = "claude-3-7-sonnet-20250219")  
chat$chat("How many n's in unconventional?")  
#> There are 2 letter "n"s in the word "unconventional".
```



# And they're not even consistent

```
chat ← chat_anthropic(model = "claude-3-7-sonnet-20250219")
```

```
chat$chat("How many n's in unconventional?")
```

```
#> There are 3 letter "n"s in the word "unconventional":
```

```
#>
```

```
#> u**n**co**n**ve**n**tional
```



# And they're not even consistent

```
chat ← chat_anthropic(model = "claude-3-7-sonnet-20250219")  
chat$chat("How many n's in unconventional?")  
#> There is 1 letter "n" in the word "unconventional".
```

ns	count
1	14
2	76
3	10



# And the goal posts are constantly moving

```
chat ← chat_anthropic()
#> Using model = "claude-sonnet-4-20250514".
chat$chat("How many n's in unconventional?")
#> To count the number of "n's" in "unconventional", I'll examine each letter:
#>
#> u-n-c-o-n-v-e-n-t-i-o-n-a-l
#>
#> Looking at each position:
#> - Position 2: n
#> - Position 5: n
#> - Position 8: n
#> - Position 12: n
#>
#> There are **4** n's in "unconventional".
```



# So LLMs kind of suck:

- The results are stochastic
- The models are constantly changing
- The difference between good results and bad results can be razor thin (⚡ jagged edge ⚡)
- They always give plausible results
- They rarely admit doubt or lack of knowledge



Yes, and



# Computers can now write poems

```
chat ← chat_anthropic()  
chat$chat("Write a limerick about the tidyverse")  
#> Here's a limerick about the tidyverse:  
#>  
#> A coder who worked with great zeal,  
#> Said "tidyverse makes data feel real!  
#> With dplyr to wrangle,  
#> And ggplot2's angle,  
#> My pipe dreams are now a good deal!"
```



And create art





# And make fun games

```
library(ellmer)
```

```
chat ← chat_anthropic(
```

```
"
```

```
  You're a quiz master who asks CHALLENGING questions about  
  the tidyverse. Make a fun game show"
```

```
)
```

```
live_browser(chat)
```



# Or more practically

```
chat ← chat_anthropic("Be terse")
```

“System” prompt

```
chat$chat("
```

What's the name of that base R function that finds names of functions that match some characters?

```
")
```

```
#> `apropos()``
```

```
#>
```

```
#> You can use it like `apropos("plot")` to find all
```

```
#> functions containing "plot" in their name.
```



# Or more practically

```
chat$chat("
  I'm the developer of http2 and I want to improve the behaviour of
  req_throttle(), which prevents requests from overwhelming a server.
  What tools, techniques, algorithms or approaches should I be
  thinking about?
")
#> Here are key approaches for improving `req_throttle()`:
#>
#> ## Rate Limiting Algorithms
#> - **Token bucket**: Allows bursts while maintaining average rate
#> - **Leaky bucket**: Smooth, constant rate limiting
#> - **Fixed/sliding window**: Time-based quotas
#> - **Exponential backoff**: Automatic retry with increasing delays
```



Two main goals of this talk:

Useful skills

Demystify jargon



1. Structured data

2. Tool calling

3. Coding



# Structured data

---



# Your turn

```
# How would you extract name and age from this data?
```

```
prompts ← list(  
  "I go by Alex. 42 years on this planet and counting.",  
  "Pleased to meet you! I'm Jamal, age 27.",  
  "They call me Li Wei. Nineteen years young.",  
  "Fatima here. Just celebrated my 35th birthday last week.",  
  "The name's Robert - 51 years old and proud of it.",  
  "Kwame here - just hit the big 5-0 this year."  
)
```



# LLMs are generally good at this sort of task

```
chat ← chat_anthropic()
chat$chat("Extract the name and age from each sentence I give you")
chat$chat(prompts[[1]])
#> **Name:** Alex
#> **Age:** 42
chat$chat(prompts[[2]])
#> **Name:** Jamal
#> **Age:** 27
chat$chat(prompts[[3]])
#> **Name:** Li Wei
#> **Age:** 19
```



# But wouldn't it be nice to get an R data structure?

```
chat$chat(prompts[[3]])
```

```
#> list(
```

```
#>   name = "Li Wei",
```

```
#>   age  = 19
```

```
#> )
```



# Well you can 😊

```
type_person ← type_object(  
  name = type_string(),  
  age = type_number()  
)
```

JSON schema

```
chat$chat_structured(prompts[[1]], type = type_person)
```

```
#> List of 2
```

```
#> $ name: chr "Alex"
```

```
#> $ age : int 42
```



# And you can do this to many prompts at once

```
parallel_chat_structured(chat, prompts, type = type_person)
```

```
#>      name age
```

```
#> 1   Alex  42
```

```
#> 2  Jamal  27
```

```
#> 3 Li Wei  19
```

```
#> 4 Fatima  35
```

```
#> 5 Robert  51
```

```
#> 6  Kwame  50
```







And you're not limited to just text...





# And it's not a lot of code

```
paths ← dir("animals", full.names = TRUE)
images ← lapply(paths, \(x) list(content_image_file(x)))

type_animal_photo ← type_object(
  animal = type_string(),
  background_colour = type_string()
)

parallel_chat_structured(chat, images, type = type_animal_photo)
```



animal	background_colour
sheep	blue
bird	pink
kitten	gray
dog	white

animal	background_colour
sheep	blue
bird	pink
cat	gray
dog	white

animal	background_colour
sheep	blue
bird	pink
cat	gray
dog	white

animal	background_colour
sheep	blue
bird	pink
cat	beige
dog	white



animal	background_colour
sheep	blue
bird	pink
kitten	gray
dog	white



animal	background_colour
sheep	blue
bird	pink
cat	gray
dog	white

animal	background_colour
sheep	blue
bird	pink
cat	beige
dog	white





<https://vitals.tidyverse.org/>



# Tool calling

---



# LLMs don't have access to live data about the world

```
chat ← chat_anthropic("Be terse")
```

```
chat$chat("What day is it today?")
```

```
#> I don't have access to real-time information, so I can't
```

```
#> tell you what day it is today. You can check your device's
```

```
#> calendar or clock for the current date.
```



# Or the ability to make changes to the world

```
chat ← chat_anthropic("Be terse")
```

```
chat$chat("Delete the csv files in my working directory")
```

```
#> ```bash
```

```
#> rm *.csv
```

```
#> ```
```

```
#>
```

```
#> This will delete all files with the `.csv` extension in your
```

```
#> current working directory.
```







# A tool is just a function with some extra metadata

```
today ← tool(  
  function() Sys.Date(),  
  name = "today",  
  description = "Get today's date",  
  arguments = list()  
)  
chat$register_tool(today)
```



# Now the model can know what day it is

```
chat$chat("What day is it today?")
```

```
#> o [tool call] today()
```

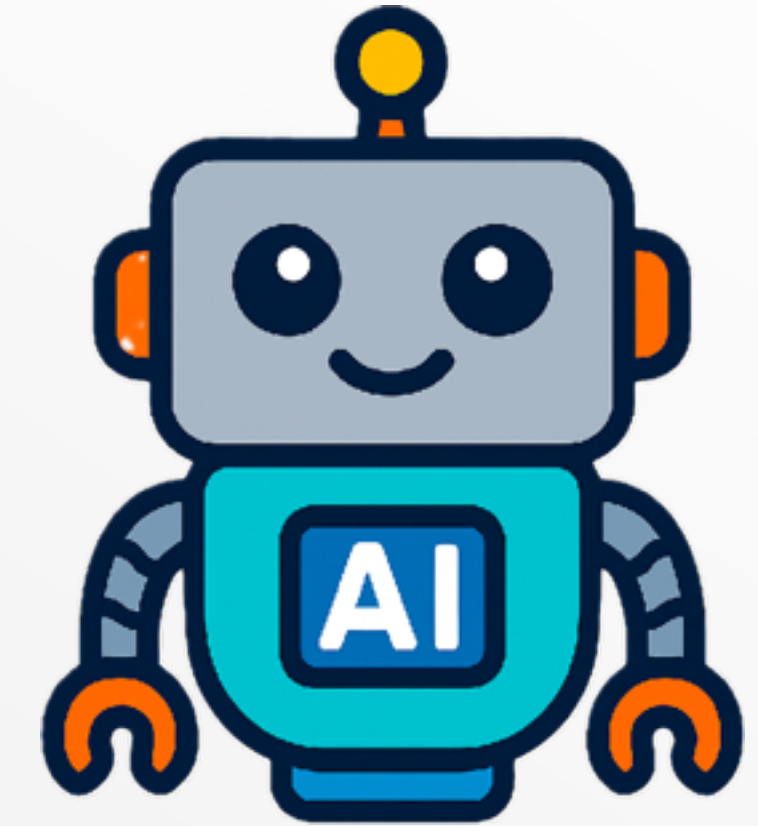
```
#> ● #> "2025-07-16"
```

```
#> Today is Wednesday, July 16, 2025.
```





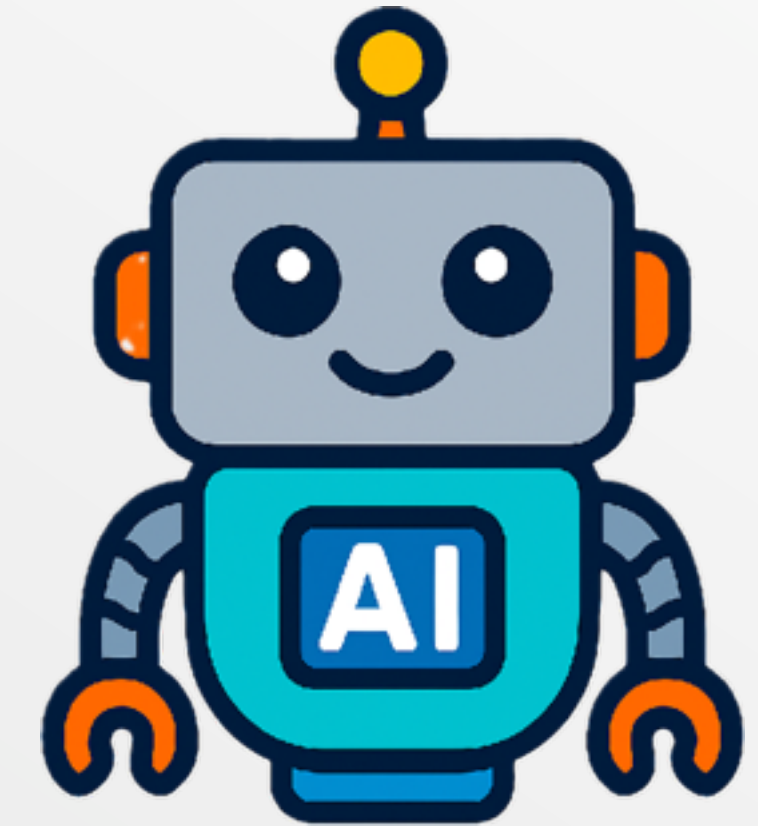
What day is it today?



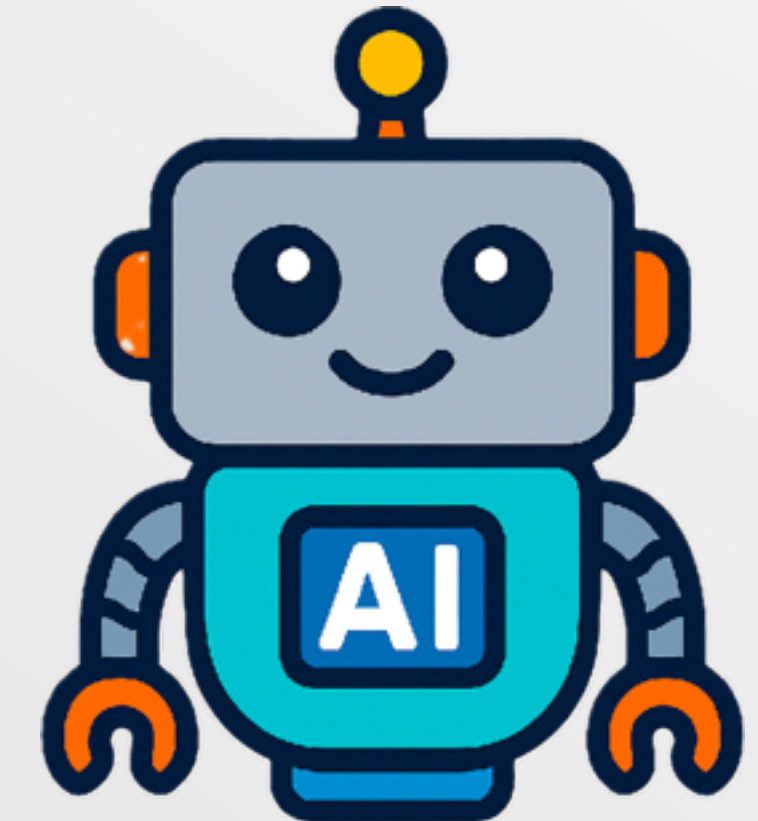
Please call today()



2025-07-16



Today is July 16, 2025. That's a Wednesday.





# You can see this if you print the chat object

chat

```
#> <Chat Anthropic/claude-sonnet-4-20250514>
```

```
#> — user [372] _____
```

```
#> What day is it today?
```

```
#> — assistant [34] _____
```

```
#> [tool request (toolu_01BdjCpkVs41AzyNGaGdPqBU)]: today()
```

```
#> — user [20] _____
```

```
#> [tool result (toolu_01BdjCpkVs41AzyNGaGdPqBU)]: "2025-07-16"
```

```
#> — assistant [15] _____
```

```
#> Today is Wednesday, July 16, 2025.
```



# We can use this to give the LLM traditional computing powers

```
chat ← chat_anthropic()
chat$register_tool(tool(
  function(x, y) format((x + 0) * (y + 0), scientific = FALSE),
  name = "multiply",
  description = "Multiply two numbers together",
  arguments = list(
    x = type_number(),
    y = type_number()
  )
))
```



# So now we get the correct result

```
chat$chat(interpolate("What's {{x}} * {{y}}?"))
```

```
#> I'll multiply those two numbers for you.
```

```
#> o [tool call] multiply(x = 20598162, y = 83106206)
```

```
#> ● #> 1711835094393372
```

```
#> The result of 20598162 * 83106206 is 1,711,835,094,393,372.
```


```
format(x * y, big.mark = ",", scientific = FALSE)
```

```
#> [1] "1,711,835,094,393,372"
```



HW What is  $20598162 \times 83106206$  ?

 Analyzed data

View analysis 

javascript 


```
// Calculate 20598162 * 83106206  
const result = 20598162 * 83106206;  
console.log("20598162 * 83106206 =", result);  
  
// Let's also verify this calculation by breaking it down  
console.log("First number:", 20598162);  
console.log("Second number:", 83106206);  
console.log("Product:", result);
```

$20598162 \times 83106206 = 1,711,835,094,393,372$



# Or we can make an agent

```
chat$register_tool(tool(  
  function() dir(),  
  name = "ls",  
  description = "Lists the files in the current directory",  
))  
chat$register_tool(tool(  
  function(path) unlink(path),  
  name = "rm",  
  description = "Delete one or more files",  
  arguments = list(  
    path = type_array(type_string())  
  )  
))
```



aka a character vector



# Now my chat bot can delete files on my computer

```
chat$chat("Delete all the csv files in the current directory")
#> I'll first check what files are in the current directory,
#> then delete any CSV files.
#> ○ [tool call] ls()
#> ● #> 1-coding-boilerplate.R
#>     #> a.csv
#>     #> b.csv
#>     #> ...
#> Now I'll delete the CSV files I found (a.csv and b.csv):
#> ○ [tool call] rm(path = c("a.csv", "b.csv"))
#> ● #> true
#> Done. Deleted `a.csv` and `b.csv`.
```







# Coding

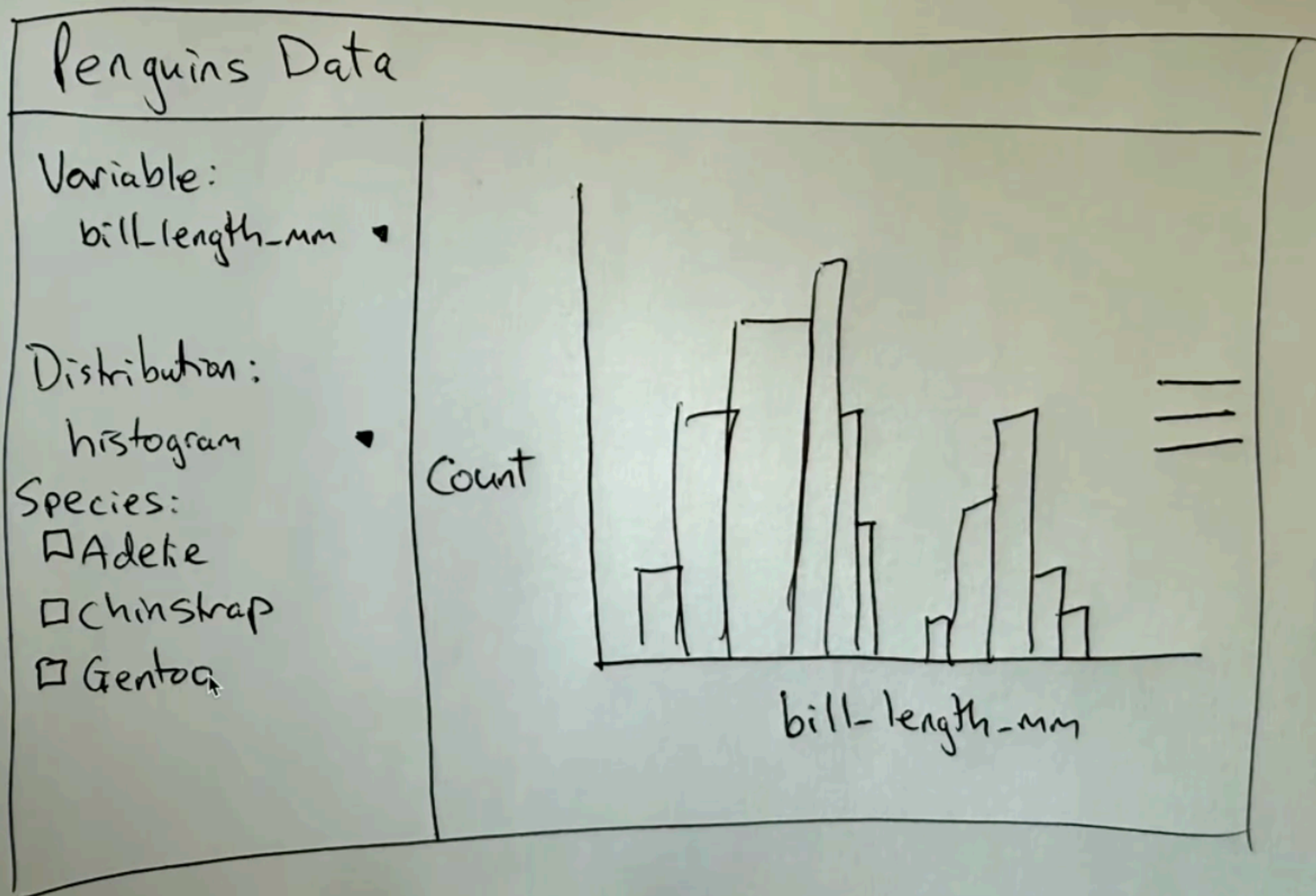
---



# Clear wins

- Amazing for quickly generating demos
  - Shiny apps
  - Example data





Example courtesy of George Stagg



# Clear wins

- Amazing for quickly generating demos
  - Shiny apps
  - Example data
- Great at translations:
  - command line curl -> httr2
  - Latex -> Quarto
  - R code -> STAN
  - JSON -> unit tests
  - SQL -> dplyr
- Explain and critiquing code



# Your turn

How do I make my code return every consecutive pair of characters, not just the first? The code below returns ab, instead of ab, cd, ef.

```
x <- "abcdef"  
start <- seq(1, nchar(x), by = 2)  
substr(x, start, start + 1)
```



# You can use tools to make a coding agent

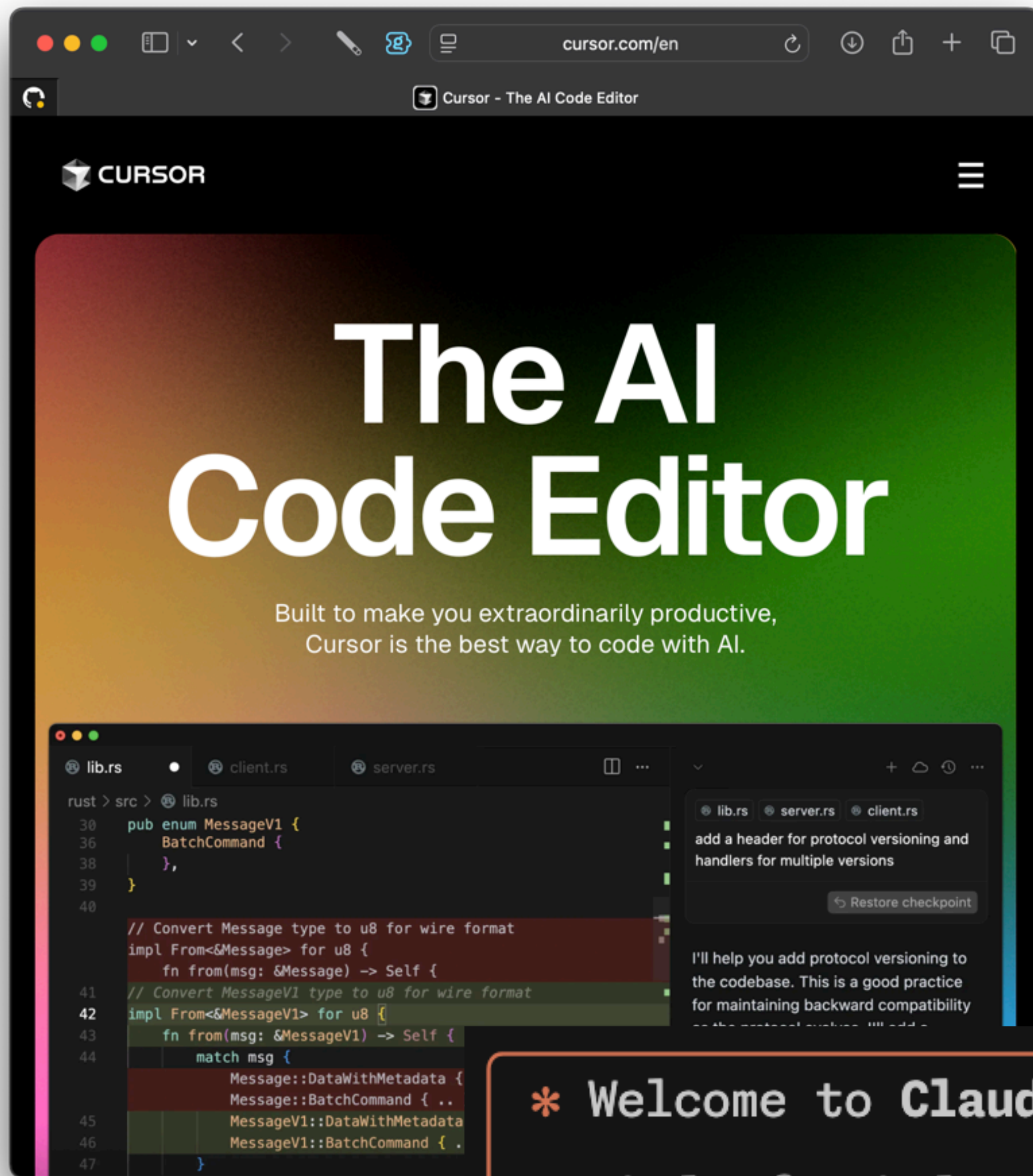
```
chat$register_tool(tool(  
  function(code) {  
    eval(parse(text = code), envir = globalenv())  
  },  
  name = "evaluate",  
  description = "Run R code",  
  arguments = list(  
    code = type_string()  
  )  
))
```



# And give it the ability to read the docs

```
chat$register_tool(tool(  
  btw::btw_tool_docs_help_page,  
  name = "read_docs",  
  description = "Get help for a function",  
  arguments = list(  
    topic = type_string("function name"),  
    package_name = type_string("Package name, if known", required =  
FALSE),  
  )  
))
```





\* Welcome to Claude Code research preview!

*/help for help, /status for your current setup*

cwd: /Users/hadleywickham/Documents/talks/25-llms-data-science



# First response

The issue is that your ``by = 2`` is skipping every other position.  
For consecutive pairs, you need ``by = 1``:

```
x ← "abcdef"
# Stop at nchar(x) - 1 to avoid going past the string
start ← seq(1, nchar(x) - 1, by = 1)
substr(x, start, start + 1)
#> "ab"
```



# Second response

Let me check what's happening:

```
x ← "abcdef"
start ← seq(1, nchar(x) - 1, by = 1)
print(start)
#> [1] 1 2 3 4 5
result ← substr(x, start, start + 1)
print(result)
#> [1] "ab"
```



# Third response

That's strange. Let me debug this further:

```
x ← "abcdef"
nchar(x)
start ← seq(1, nchar(x) - 1, by = 1)
print(start)
#> [1] 1 2 3 4 5
end ← start + 1
print(end)
#> [1] 2 3 4 5 6
result ← substr(x, start, end)
print(result)
#> [1] "ab"
```



# Final response

I see the issue now! ``substr()`` is vectorized but returns a single string when given vectors. Use ``substring()`` instead:

```
x ← "abcdef"
start ← seq(1, nchar(x) - 1, by = 1)
result ← substring(x, start, start + 1)
print(result)
[1] "ab" "bc" "cd" "de" "ef"
```



# Conclusion

---



# LLM concerns

- **Cost/equality of access:** \$5 on Claude gets you pretty far; Gemini has a generous free tier. Local models not worth it IMO.
- **Environmental concerns:** worth considering, but small and decreasing on individual level. More impactful ways to save energy/water.
- **Data privacy:** legit as individual. Not a problem for most bigger orgs as most data already lives in some cloud, and cloud providers run LLMs.
- **Replacing artists:** Definite risk at societal level. Personally, trying to supplement, not replace.
- **Evil billionaires:** Eat the rich?



1. Structured data

2. Tool calling

3. Coding