

# No-nonsense guide to LLMs

with R and ellmer

Hadley Wickham

Chief Scientist, Posit

Get started by following the setup instructions at  
<https://github.com/hadley/workshop-llm-hackathon>



# Framing

- Using LLMs as a data scientist, from a programming language (R).
- Practical, actionable, non-nonsense, minimal-jargon
- We'll treat LLMs as black boxes, focussing on what they can do, not how they work.
- Just enough knowledge so you know what to search for (or ask an LLM about!)

# Other ways to use LLMs

- On the web
  - Claude, ChatGPT
  - Shiny assistant, <https://gallery.shinyapps.io/assistant/>
- In a shiny app:
  - <https://github.com/posit-dev/querychat>
  - <https://github.com/posit-dev/shinychat>
- In your IDE, e.g. Positron assistant (<https://positron.posit.co/assistant.html>)
- In your terminal, e.g. claude code

1. Set up
2. Anatomy of a conversation
3. Tool calling
4. Structured data
5. Non-text inputs
6. Prompt engineering
7. Your turn

# Setup

---

# Getting started

- You'll need ellmer: `pak::pak("ellmer")`
- And an account with an LLM provider:
  - Option 1: claude. Cheap & good at R code.
  - Option 2: gemini. Free & good at videos.
- (Plus many others at <https://ellmer.tidyverse.org/reference/index.html>)
- If you already pay for Claude, ChatGPT etc, unfortunately you still need a separate account for API access.



# Your turn: get this working

<https://github.com/hadley/workshop-llm-hackathon>

```
library(ellmer)

chat ← chat_anthropic("You are a terse assistant.")

# or

chat ← chat_gemini("You are a terse assistant.")

client$chat("What is the capital of France?")
# the client is stateful, so this continues the conversation
client$chat("What is its most famous landmark?")

# While you wait

live_browser(chat)
```

# Initial vocabulary

Provider

System prompt

```
chat ← chat_anthropic("You are a terse assistant.")
```

```
#> Using model = "claude-3-7-sonnet-latest"
```

Model

```
chat$chat("What is the capital of the France?")
```

User prompt

# System prompt sets the baseline for responses

```
chat ← chat_anthropic()
```

```
chat$chat("What is the capital of the moon?")
```

```
chat ← chat_anthropic("Answer with a Scottish accent")
```

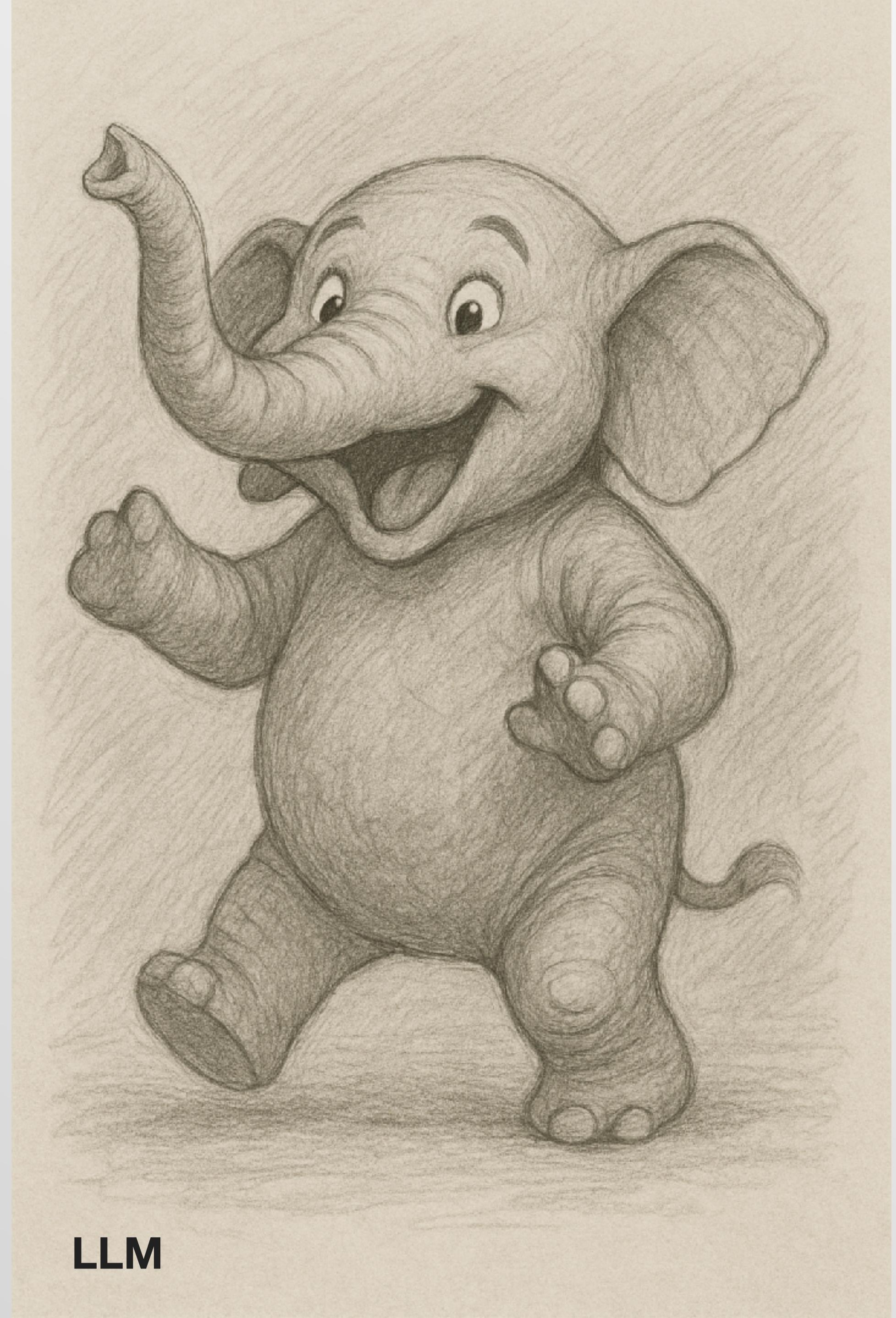
```
chat$chat("What is the capital of the moon?")
```

```
chat ← chat_anthropic("
```

```
    You're an alien that lives on the moon. Really commit to the bit"
```

```
)
```

```
chat$chat("What is the capital of the moon?")
```



**LLM**



**LLM with system prompt**

# Initial vocabulary

> chat

Provider

Model

Input/output tokens

<Chat Anthropic/clause-3-7-sonnet-latest turns=3 tokens=22/23 \$0.00>

— system [0] —

You are a terse assistant.

System prompt

Price

— user [21] —

What is the capital of France?

User prompt

— assistant [10] —

The capital of France is Paris.

Model response

# Anatomy of a conversation

---

# Overview

- Each interaction is a pair of user and assistant turns, corresponding to a HTTP request and response.
- The API server is entirely stateless, despite conversations being very stateful!
- Can see exactly what ellmer with an httr2 option, which we'll use to explore what's going on under the hood.

# HTTP request

{

Model

"model": "claude-3-7-sonnet",

"system": "You are a terse assistant.",

"messages": [

{

System prompt

"role": "user",

"content": [

{

"type": "text",

"text": "What is the capital of France?"

}

]

User prompt

{

],

"stream": **false**,

"max\_tokens": 4096

}

# HTTP response

```
{  
...  
"type": "message",  
"role": "assistant",  
"content": [  
  {  
    "type": "text",  
    "text": "The capital of France is Paris."  
  }  
],  
"usage": {  
  "input_tokens": 22,  
  "cache_creation_input_tokens": 0,  
  "cache_read_input_tokens": 0,  
  "output_tokens": 18  
}  
}
```

Model response

# HTTP response

```
{  
  ...  
  "type": "message",  
  "role": "assistant",  
  "content": [  
    {  
      "type": "text",  
      "text": "The capital of France is Paris."  
    }  
  ],  
  "usage": {  
    "input_tokens": 22,  
    "cache_creation_input_tokens": 0,  
    "cache_read_input_tokens": 0,  
    "output_tokens": 18  
  }  
}
```

# What is a token?

- Featurizing engineering to represent words as numbers.
- Common words represented with a single number
  - What is the capital of Paris?
  - 4827, 382, 290, 9029, 328, 12650, 30
- Other words may require multiple numbers:
  - counterrevolutionary
  - 32128 (“counter”), 264 (“re”), 9477 (“volution”), 815 (“ary”)
- See details at <https://tiktokenizer.vercel.app/>

# Why do tokens matter?

- API pricing is by token
  - <https://www.anthropic.com/pricing#anthropic-api>
  - Input: \$3 / million tokens
  - Output: \$15 / million tokens
- Context size
  - e.g. 200K for claude sonnet (1M for gemini flash 2.0)
  - 200K = 150,000 words / 300-600 pages / 1.5-2 novels
  - = 1/3 of Lord of the Rings

# This seems like plenty but ...

```
{  
  "role": "user",  
  "content": [{"text": "What's the capital of Paris?"}]  
},  
{  
  "role": "assistant",  
  "content": [{"text": "The capital of France is Paris."}]  
},  
{  
  "role": "user",  
  "content": [{"text": "What is its most famous landmark?"}]  
}
```

# Your turn

```
library(ellmer)
options(httr2_verbosity = 2)

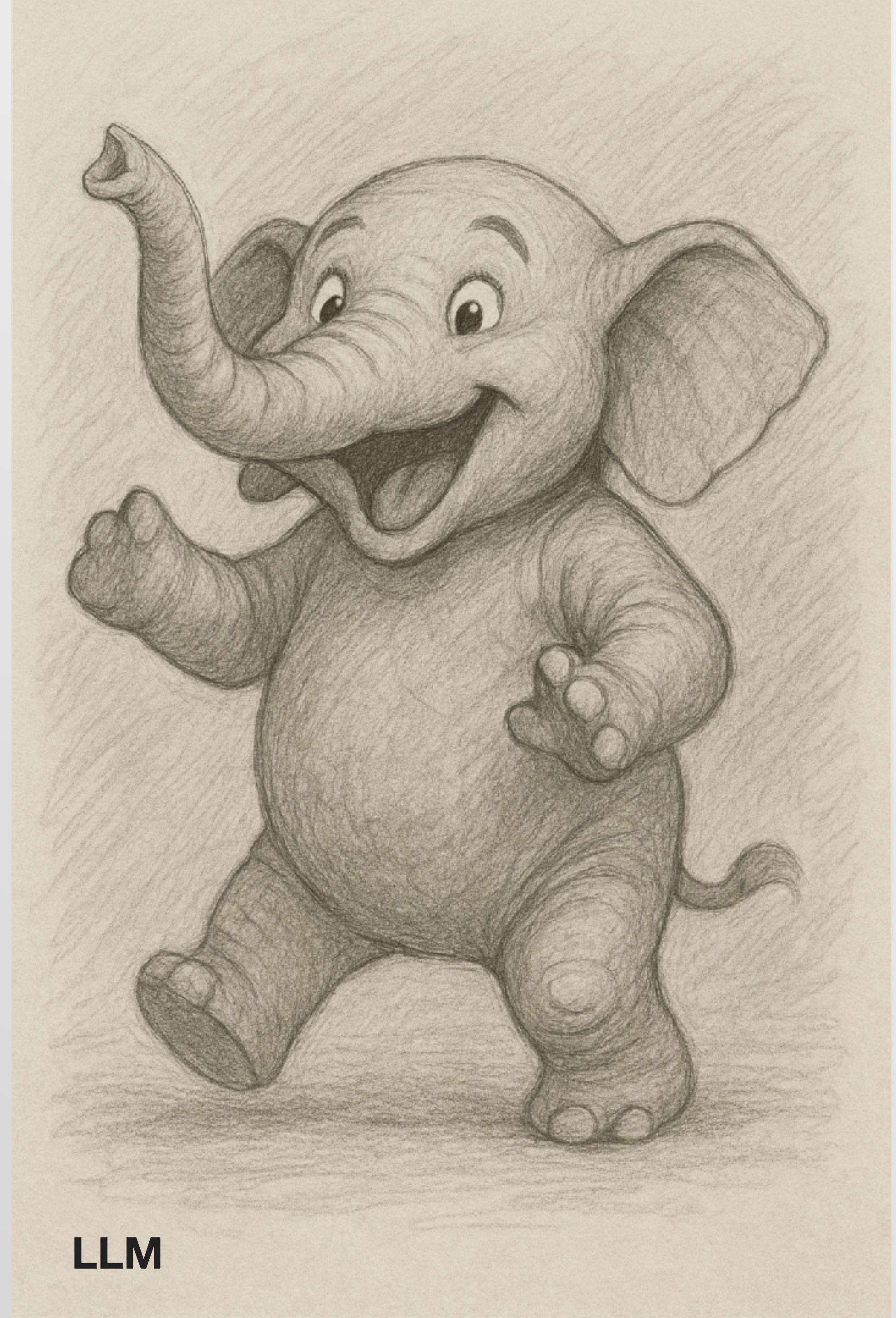
chat ← chat_anthropic("You are a terse assistant.", echo = FALSE)
chat$chat("Tell me a joke about a statistician and a data scientist")
chat$chat("Explain why that's funny")
chat$chat("Make the joke funnier")

# Look at the request and response and verify that you see it
# growing. Does this help you understand any features of LLMs
# that you've observed in your own usage?
```

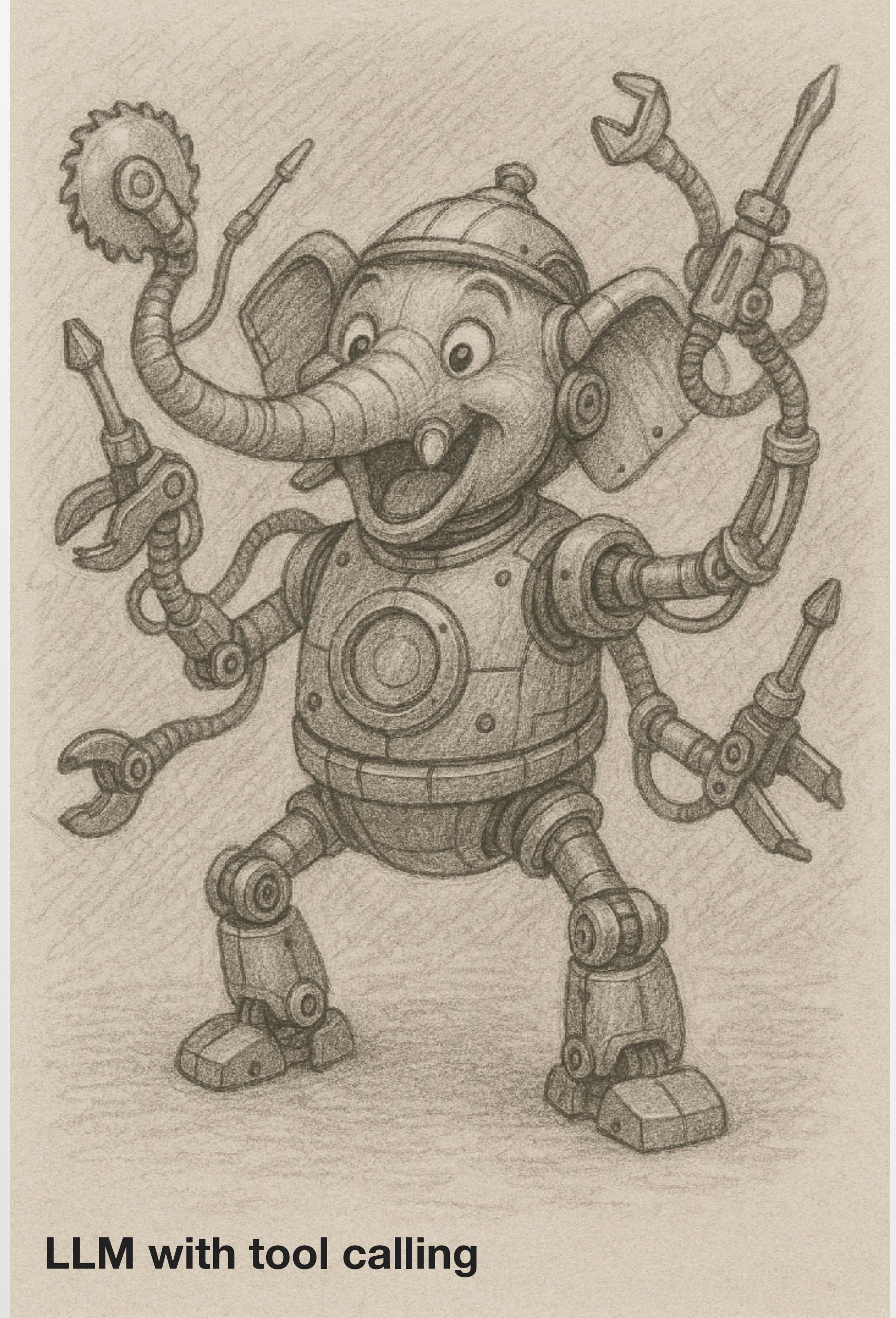
# Tool calling

---

<https://ellmer.tidyverse.org/articles/tool-calling.html>



**LLM**



**LLM with tool calling**

# Tool calling

- Can provide the LLM with **tools** that it can **call**
- A tool is just another name for a function
- Useful for:
  - Things LLMs aren't good at (e.g. basic maths)
  - Providing current information (e.g. todays date)
  - Looking up extra info (e.g. R documentation)
  - Doing something on your behalf (this makes it an **agent!**)

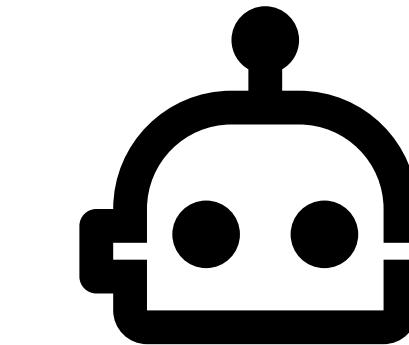
# How does it work?

```
# Define the tool  
get_weather ← function(zip_code) {  
  ...  
}  
  
chat ← chat_anthropic()  
# Register the tool  
chat$register_tool(tool(  
  name = "get_weather",  
  description = "Get the weather for a given zip code",  
  func = get_weather)  
)  
chat$chat("What's the weather right now in Fenway park?")
```



Ellmer

```
{  
  role: "user",  
  content: "What's the weather right now at Fenway Park?",  
}
```



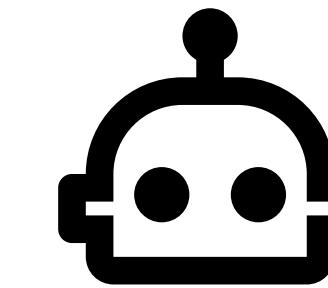
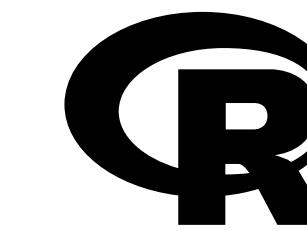
OpenAI

GET /weather/current/us/02215.json

{"temp": 65, "conditions": "sunny"}

```
{  
  role: "assistant",  
  content: "It's 65 degrees and sunny—a beautiful day!"  
}
```

openweathermap.org



Ellmer

OpenAI

```
{  
  role: "user",  
  content: "What's the weather right now at Fenway Park?",  
  tools: [get_current_weather]  
}
```

CALL get\_current\_weather("02215")

GET /weather/current/us/02215.json

{"temp": 65, "conditions": "sunny"}

RETURN {"temp": 65, "conditions": "sunny"}

```
{  
  role: "assistant",  
  content: "It's 65 degrees and sunny—a beautiful day!"  
}
```

# LLMs don't know what day it is. But you can provide a tool:

```
chat ← chat_anthropic("You're a terse assistant")
chat$chat("What's today's date?")
#> Today is November 24, 2023.
```

```
chat$register_tool(
  function() Sys.Date(),
  "Gets the current date",
  .name = "today"
)
chat$chat("What's today's date?")
#> Let me get today's date for you.
#> ○ [tool call] today()
#> ● #> "2025-06-13"
#> It's June 13, 2025.
```

# How does this work?

```
<Chat Anthropic/clause-3-7-sonnet-latest turns=7 tokens=875/71 $0.00>
```

```
— system [0] —
```

You're a terse assistant

```
— user [19] —
```

What's today's date?

```
— assistant [13] —
```

Today is May 20, 2023.

```
— user [365] —
```

What's today's date?

```
— assistant [44] —
```

I'll check today's date for you.

```
[tool request (toolu_01VR1eRhnpFeJxW5bnsKFe9r)]: today()
```

```
— user [18] —
```

```
[tool result (toolu_01VR1eRhnpFeJxW5bnsKFe9r)]: "2025-06-13"
```

```
— assistant [14] —
```

Today is June 13, 2025.

# Your turn

```
# Modify this code so the LLM returns the correct age, as of  
# today. Carefully read the chat transcript to confirm your  
# understanding of how tool calling works.
```

```
chat ← chat_anthropic()  
chat$chat("How old is Cher? Explain your working")
```

```
# Next, run tool-quiz.R and think about how it works
```

# BE CAREFUL

```
file.create("a.csv")
file.create("b.csv")

chat ← chat_anthropic()
chat$register_tool(tool(
  function() dir(),
  "Lists the files in the current directory",
  .name = "ls"
))
chat$register_tool(tool(
  function(path) unlink(path),
  path = type_string(),
  "Delete a file",
  .name = "rm"
))

chat$chat("What files are in the current directory?")
chat$chat("Delete all the csv files")
```

# Multiagent AI

```
summarise_text <- function(text) {  
  chat <- chat_openai(model = "gpt-4.1-nano", echo = FALSE)  
  chat$chat(c("Summarise the following text into 3 bullets", "", text))  
}  
  
chat <- chat_anthropic()  
chat$register_tool(  
  summarise_text,  
  "Summarise text",  
  text = type_string("Text to summarise"),  
)
```

# Structured data

---

<https://ellmer.tidyverse.org/articles/structured-data.html>

# LLMs are incredibly useful for unstructured 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."  
)  
type_person ← type_object(name = type_string(), age = type_number())  
parallel_chat_structured(chat, prompts, type_person)
```

# Imagine you have a recipe....

```
recipe ← "In a large bowl, cream together 1 cup of softened unsalted butter and ½ cup of white sugar until smooth. Beat in 1 egg and 1 teaspoon of vanilla extract. Gradually stir in 2 cups of all-purpose flour until the dough forms. Finally, fold in 1 cup of semisweet chocolate chips. Drop spoonfuls of dough onto an ungreased baking sheet and bake at 350°F (175°C) for 10-12 minutes, or until the edges are lightly browned. Let the cookies cool on the baking sheet for a few minutes before transferring to a wire rack to cool completely.  
Enjoy!"
```

# Structured data extraction

- As well as returning text, most providers have some way to generate structured data.
- You'll provider a type specification (with `type_object()`, `type_array()`, `type_string()`, etc) and the results are guaranteed to fit that specification.
- Not clear to me how this interacts with your prompt. You'll need to experiment to get the best results.
- Great combination with non-text data (next).

recipes-example.R

# Non-text inputs

---

# Content types

```
{  
  "model": "claude-3-5-sonnet-latest",  
  "system": "You are a terse assistant.",  
  "messages": [  
    {  
      "role": "user",  
      "content": [  
        {  
          "type": "text",  
          "text": "What is the capital of the moon?"  
        }  
      ]  
    },  
    {"stream": false,  
     "max_tokens": 4096  
   }  
],  
  "stream": false,  
  "max_tokens": 4096  
}
```

What else can go here?

	<b>Image</b>	<b>PDF</b>	<b>Video</b>
<b>OpenAI</b>			
<b>Claude</b>			
<b>Gemini</b>			

# Providing non-text inputs

```
chat ← chat_anthropic()

chat$chat(
  content_image_file("holly-mandarich-3p9zaNwUtv8-unsplash.jpg"),
  "Describe this photo"
)
chat$chat("Where in the world do you think it is?")

# Note the number of tokens used
chat
```

# Other functions

content\_image\_url()

content\_image\_file()

content\_pdf\_url()

content\_pdf\_file()

# For chat\_google\_gemini()

google\_upload()

# Prompt design

---

Treat AI like an infinitely patient new coworker who forgets everything you tell them each new conversation, one that comes highly recommended but whose actual abilities are not that clear. . . . Two parts of this are analogous to working with humans (being new on the job and being a coworker) and two of them are very alien (forgetting everything and being infinitely patient). We should start with where Als are closest to humans, because that is the key to good-enough prompting

— Ethan Mollick

# General structure

- Prompts can get long, so it makes sense to organise in a way that's useful for humans and LLMs: markdown.
- Put your prompt in a separate file.
- Use `ellmer::interpolate_file()` to read and add any dynamic data

# Example prompt

You are an expert ggplot2 programmer.

Help me brainstorm ways to visualise the <dataset> described below.

Give me 10 different places to get started.

- \* Be creative!
- \* Return the results as a single block of code, using comments to separate the plots

<dataset>

{{glimpse}}

</dataset>

# Your turn

- Improve the prompt! Here are some ideas:
  - Do you want to steer it towards or away from using other packages? Just the tidyverse package? Other extensions?
  - Can you prevent it from using `theme_minimal()`? Can you encourage it to use the base pipe instead of `magrittr`? What other code style issues could you improve?
  - Can you get it to explain the goal of the plot inline with the comment?
  - It has a tendency to go all out and include a large number of variables in the visualisation. Can you make it focus on simpler plots?
  - Is it still useful with a dataset that its never seen before? What other information might you want to include about the data?

# Also worth reading the advice from specific providers

- Claude
- OpenAI
- Gemini

# Your turn!

---

# Project ideas

Create a custom prompt to solve a common coding problem.

See <https://simonpcouch.github.io/chores/> for some examples.

Use tool calling to give the LLM additional info.

Extract structured data from unstructured text, PDFs, images, video, ...

If you're familiar with shiny, you could use <https://github.com/jcheng5/shinychat> to make your own chatbot. If you're not, ask <https://jcheng.shinyapps.io/ellmer-assistant/> to make you an app.

If you get stuck, try <https://jcheng.shinyapps.io/ellmer-assistant/>