LLM via API

OpenRouter is a service that provides access to many LLMs through a web app and an API. Some are paid, others are free up to a certain number of queries. Follow the OpenRouter guide (under Tools on the course website) to create an account and an API key with a $0 limit, so you use only the free models, and to set the `OPENROUTER_API_KEY` environment variable in your terminal and in R. Run the check at the end of the guide before you start.

1. Click on the "Models" tab to see the available models. How many total models are there? How many are free to use?[1]

2. Search for and then click on a model called "Google: Gemma 4 31B (free)" and read the documentation page. What are its capabilities?

Everyone using the free version of Gemma 4 31B shares it, and when it's busy, a request fails with `HTTP 429 Too Many Requests`. If that happens, wait a minute and try again. If it keeps happening, use another free model that can call tools, such as `nvidia/nemotron-3.5-super-vl:free`, in place of Gemma for the rest of this problem set. Its answers will be worded differently from the examples in the solutions, and it may print its reasoning in a `<thinking>` block before each answer.

At the bottom of the page, it shows example API calls in several programming languages. Copy the `curl` example and paste it into a JSON file in Positron (create a new file and give it a `.json` file extension). You can select all of the contents of that file, right click it, and select "Format Document" to make it more readable.

3. What is the endpoint URL for the API call?

[1] Use the filters available in the left sidebar. Note that some models are free up to a certain number of queries per day, so be judicious.

4. "Headers" (indicated with `-H`) are similar to parameters included in the URL of a GET request. What are the names of the headers that you will pass and what are their values?

5. The data that your computer sends with the request is flagged with `-d` and is sent as a JSON object. What are the names of the keys in this object?

6. Run the `curl` command in your terminal just as you copied it, in a terminal where `OPENROUTER_API_KEY` is set. The shell replaces `$OPENROUTER_API_KEY` with your key before the request is sent[2]. Copy and paste the response in your JSON file and format the document again to make it easier to read. What is the name of the key in the response that contains the text generated by the model?

7. List 5 other keys and values that are present in the response besides the text generated by the model.

8. In the space below, write a new `curl` command that only sends the prompt "The capital of Canada is".

[2] If the response is an authentication error, the variable is not set in this terminal. See "Check that it worked" in the OpenRouter guide.

9. Send the `curl` command. Beyond the content of the text generated by the model, how does the response compare to the previous one?

10. Return to OpenRouter and click on the "Chat" tab. Select the "Google: Gemma 4 31B (free)" model and enter the same prompt ("The capital of Canada is") in the text box. How does the response compare to the previous one?

---

11. Create an R script and load `ellmer`. Create a chat object using `chat_openrouter()` and save it to `or1`. You will need to pass the model name as an argument (it is the same as in the `curl` request). Like `chat_claude()` in lecture, `chat_openrouter()` reads your key from the environment variable. Write your code below then run it.

12. Use `or1` to send the prompt "The capital of Canada is". What is the response?

(Optional but cool): Most models have a standard set of other parameters that you can send that will determine how the model generates its text response. You can explore these by reading `?chat_openrouter()`. Max tokens is the maximum length of the response (in tokens, where 1 token is roughly 4 characters of text). Temperature controls how "creative" the model is allowed to be: a temperature of 0 means it will always pick the most likely next token, while values closer to 2 will make it more likely to pick less probable tokens, resulting in more varied and creative responses. You pass these parameters as a named list to the `params` argument of `chat_openrouter()`.

13. Create a new chat object called `or3` that uses the "google/gemma-4-31b-it:free" model but sets the `temperature` parameter to 2 and the `max_tokens` parameter to 500. Use `or3` to send the prompt "Describe new types of punctuation and the emotions they evoke". Initialize `or3` again and send the same prompt several times to get a sense of how the responses vary. Change the temperature of `or3` to 0 and send the same query several times. Write your observations below.

Completion and Context

These problems continue with `or1` from the last section, which you asked for the capital of Canada. If you've started a new R session, set the key again with `Sys.setenv()` at the console, then create `or1` again and resend that prompt.

14. Use `or1` to send a second prompt: "What is the most popular sport?". Write the gist of the response below.

15. Create a second chat object called `or2` using the same model. Send the same prompt: "What is the most popular sport?". How does the response compare to the one from `or1`?

16. Print `or1` and `or2` at the console. How does this explain the difference in responses?

---

17. Look again at the printout of `or1`, which holds two prompts and their answers. Each assistant turn is labeled with `input=` and `output=`, counted in tokens. Which turn had the larger `input`? Explain why, using what the client sends with each request.

18. Without running it, suppose you send `or1` a third prompt: "Where is it played the most?". List the `role` of every message in the body of that request, in order.

19. Send that prompt to `or1`, and send the same prompt to a new chat object, `or4`. Which sport does each response discuss? Explain the difference using the word *context*.

---

20. Create a chat object called `terse` with `chat_openrouter()` that uses the same model and a `system_prompt` telling the model to answer in five words or fewer. Write your code below.

21. Send `terse` the prompt "What is the most popular sport?" and then print `terse`. Where does the system prompt appear among the turns? Did the model follow it?

22. Write a system prompt for a chat that you would want to use while working on a problem set in this course: one that helps you learn the material instead of doing the work for you. It should state at least two conventions of the R code we write in this class. Write the system prompt below.

23. Create a chat object with your system prompt and ask it for help with a problem from an earlier problem set that you found hard. Describe one way the response followed your system prompt and one way it did not.

The code below makes a small data frame of quiz scores.

```
library(tibble)
scores <- tibble(student = c("Ana", "Ben", "Chen", "Dee", "Eli"),
                 quiz1   = c(8, 6, 9, NA, 7),
                 quiz2   = c(9, 7, 7, 8, 10))
```

24. Create a new chat object, `or5`, and send it the prompt "Which student in my data improved the most from quiz 1 to quiz 2?". What does the model say? Why can't it answer?

25. Send `or5` the same question again, this time with the data in the context. Use `format_csv()` as in lecture. Write your code and the model's answer below.

26. Answer the question yourself with a `dplyr` pipeline. Write your code and its result below. Does it agree with the model?

27. Dee has no score for quiz 1. What did the model do with Dee, and what did your pipeline do? Which one told you that it had made a choice?

(Optional, for those with code completion set up in Positron or RStudio)

28. In an empty R script, type the comment `# a function that returns the mean and sd of a numeric vector as a named list` and start a new line. Before accepting any suggestion, write down the first line that you expect. Then accept the suggestions until the function is complete. Where did the suggestion differ from what you expected, and is the suggested function correct when the vector contains an `NA`?

Tool Calls

These problems use `chat_openrouter()` with the same free model as the last two sections, `google/gemma-4-31b-it:free`, which supports tools. If Gemma is busy, select another free model to use such as `nvidia/nemotron-3.5-super-vl:free`. If you've started a new R session, check that your key is set (see "Check that it worked" in the OpenRouter guide) and load `ellmer`.

In lecture we offered the model a function that predicts a penguin's body mass from its flipper length:

```
predict_mass <- function(flipper) {
  new_penguin <- tibble(flipper_length_mm = flipper)
  predict(penguin_model, new_penguin)
}

mass_tool <- tool(
  predict_mass,
  name = "predict_mass",
  description = "Predicts a penguin's body mass, in grams, from its flipper length.",
  arguments = list(
    flipper = type_number("Flipper length, in mm.")
  )
)
```

Here is a chat that has `mass_tool` registered, printed after one call to `$chat()`:

```
<Chat Anthropic/claude-haiku-5-5 turns=4 input=947 output=83>
── user ────────────────────────────────────────────────────────────
How heavy is a penguin with 210 mm flippers? Answer in one sentence.
── assistant [input=443 output=51] ─────────────────────────────────
[tool request (toolu_015GQa...)]: predict_mass(flipper = 210L)
── user ────────────────────────────────────────────────────────────
[tool result  (toolu_015GQa...)]: 4653.1376
── assistant [input=504 output=32] ─────────────────────────────────
A penguin with 210 mm flippers is predicted to weigh about 4,653 grams.
```

29. For each of the four turns, say who wrote it: the person at the keyboard, the model, or `ellmer` (which coordinates the tools). Then say how many requests this one call to `$chat()` sent to the API, and where `4653.1376` was computed. Last, the first request had `input=443` tokens, though its question is only about 15 tokens long. What else did the client send with it?

30. Create a chat object, `or6`, with no tools, and send it the prompt "Roll two six-sided dice and tell me the total." What does the response say? Where did its numbers come from?

31. Write a function, `roll_dice()`, with arguments `n` (how many dice) and `sides` (how many sides each die has), that returns the number on each die. Use `sample()`, and call it to roll two six-sided dice. Then write `dice_tool`, a `tool()` that offers `roll_dice()` to a model, with a name, a description, and a description and type for each argument.

32. Create a chat object, `dice`, register `dice_tool`, and send it "Roll two six-sided dice and tell me the total." Print `dice`. Which turn holds the numbers that R rolled, and does the model's total match them?

33. Without running it yet, suppose you now send `dice` the prompt "Roll them again." How many requests will that call to `$chat()` send, if the model calls the tool once? List the `role` of each message in the *last* request, in order. Then run it and print `dice` to check the order of the turns.

In APIs II you wrote requests to a `/babynames` endpoint that took a name and a year. A tool can do the same job.

34. Write a function, `name_count()`, with arguments `babyname` and `babyyear`, that returns the number of babies given that name in that year in the `babynames` data, girls and boys together. Check it: how many babies were named Taylor in 1993?

35. Write `name_tool` for `name_count()`, register it with a new chat object, `names_chat`, and ask "How many babies were named Taylor in 1993, and how many in 2013?" Print the chat. What tool calls did the model make, and do its numbers agree with `name_count()`?

---

36. *Challenging*: In APIs I you requested estimated departures from BART's API and unnested them into one row per train. Write a function, `next_trains()`, that takes a station's abbreviation (such as `"DBRK"`) and returns the destination and minutes until departure of each train, as text from `format_csv()`. Offer it to a chat as a tool, then ask the chat "When does the next train leave Downtown Berkeley for Richmond?" Check its answer against your function's output.