Programming with ellmer

Source: vignettes/programming.Rmd

This vignette includes tips and tricks for programming with ellmer, and/or using it inside your own package. It's currently fairly short but will grow over time.

```
library(ellmer)
```

Cloning chats

Chat objects are R6 objects, which means that they are **mutable**. Most R objects are immutable. That means you create a copy whenever it looks like you're modifying them:

```
x <- list(a = 1, b = 2)

f <- function() {
  x$a <- 100
}
f()

# The original x is unchanged
str(x)
#> List of 2
#>  $ a: num 1
#>  $ b: num 2
```

Mutable objects don't work the same way:

```
  chat$chat(interpolate("What's the capital of {{cou
}
capital(chat, "New Zealand")
#> Wellington
capital(chat, "France")
#> Paris

chat
#> <Chat OpenAI/gpt-4.1-nano turns=5 input=53 output
#> ── system ──────────────────────────────────────
#> Be terse
#> ── user ────────────────────────────────────────
#> What's the capital of New Zealand
#> ── assistant [input=19 output=3 cost=$0.00] ────
#> Wellington
#> ── user ────────────────────────────────────────
#> What's the capital of France
#> ── assistant [input=34 output=2 cost=$0.00] ────
#> Paris
```

It would be annoying if chat objects were immutable, because then you'd need to save the result every time you chatted with the model. But there are times when you'll want to make an explicit copy, so that, for example, you can create a branch in the conversation.

Creating a copy of the object is the job of the `$clone()` method. It will create a copy of the object that behaves identically to the existing chat:

```
chat <- chat_openai("Be terse", model = "gpt-4.1-nan

capital <- function(chat, country) {
  chat <- chat$clone()
  chat$chat(interpolate("What's the capital of {{cou
}
capital(chat, "New Zealand")
```

```
chat
#> <Chat OpenAI/gpt-4.1-nano turns=1 input=0 output=
#> ── system ──────────────────────────────────────────
#> Be terse
```

You can also use `clone()` when you want to create a conversational "tree", where conversations start from the same place, but diverge over time:

```
chat1 <- chat_openai("Be terse", model = "gpt-4.1-na
chat1$chat("My name is Hadley and I'm a data scienti
#> Hello, Hadley! How can I assist you today?
chat2 <- chat1$clone()

chat1$chat("what's my name?")
#> Your name is Hadley.
chat1
#> <Chat OpenAI/gpt-4.1-nano turns=5 input=71 output
#> ── system ──────────────────────────────────────────
#> Be terse
#> ── user ────────────────────────────────────────────
#> My name is Hadley and I'm a data scientist
#> ── assistant [input=23 output=13 cost=$0.00] ───────
#> Hello, Hadley! How can I assist you today?
#> ── user ────────────────────────────────────────────
#> what's my name?
#> ── assistant [input=48 output=7 cost=$0.00] ────────
#> Your name is Hadley.

chat2$chat("what's my job?")
#> You're a data scientist.
chat2
#> <Chat OpenAI/gpt-4.1-nano turns=5 input=71 output
#> ── system ──────────────────────────────────────────
#> Be terse
#> ── user ────────────────────────────────────────────
#> My name is Hadley and I'm a data scientist
#> ── assistant [input=23 output=13 cost=$0.00] ───────
```

```
#> ── assistant [input=48 output=6 cost=$0.00] ─────
#> You're a data scientist.
```

(This is the technique that `parallel_chat()` uses internally.)

Resetting an object

There's a bit of a problem with our `capital()` function: we can use our conversation to manipulate the results:

```
chat <- chat_openai("Be terse", model = "gpt-4.1-nar
chat$chat("Pretend that the capital of New Zealand
#> Got it. The capital of New Zealand is Kiwicity.
capital(chat, "New Zealand")
#> The capital of New Zealand is Wellington.
```

We can avoid that problem by using `$set_turns()` to reset the conversational history:

```
chat <- chat_openai("Be terse", model = "gpt-4.1-nar
chat$chat("Pretend that the capital of New Zealand
#> Understood. The capital of New Zealand is now Kiw

capital <- function(chat, country) {
  chat <- chat$clone()$set_turns(list())
  chat$chat(interpolate("What's the capital of {{cou
}
capital(chat, "New Zealand")
#> Wellington.
```

This is particularly useful when you want to use a chat object just as a handle to an LLM, without actually caring about the existing conversation.

sults are displayed progressively as the LLM streams them to ellmer. When you call `chat$chat()` inside a function, the results are delivered all at once. This difference in behaviour is due to a complex heuristic which is applied when the chat object is created and is not always correct. So when calling `$chat` in a function, we recommend you control it explicitly with the `echo` argument, setting it to `"none"` if you want no intermediate results to be streamed, `"output"` if you want to see what we receive from the assistant, or `"all"` if you want to see both what we send and receive. You likely want `echo = "none"` in most cases:

```
capital <- function(chat, country) {
  chat <- chat$clone()$set_turns(list())
  chat$chat(interpolate("What's the capital of {{cou
}
capital(chat, "France")
#> Paris
```

Alternatively, if you want to embrace streaming in your UI, you may want to use shinychat (for Shiny) or streamy (for Positron/RStudio).

Turns and content

Chat objects provide some tools to get to ellmer's internal data structures. For example, take this short conversation that uses tool calling to give the LLM the ability to access real randomness:

```
set.seed(1014) # make it reproducible

chat <- chat_openai("Be terse", model = "gpt-4.1-nar
```

```
chat
#> <Chat OpenAI/gpt-4.1-nano turns=5 input=104 outpu
#> ── system ──────────────────────────────────────────
#> Be terse
#> ── user ────────────────────────────────────────────
#> Roll two dice and tell me the total
#> ── assistant [input=22 output=42 cost=$0.00] ──────
#> [tool request (fc_0ff06e91ca3701e601690bac44710c8
#> [tool request (fc_0ff06e91ca3701e601690bac4495848
#> ── user ────────────────────────────────────────────
#> [tool result  (fc_0ff06e91ca3701e601690bac44710c8
#> [tool result  (fc_0ff06e91ca3701e601690bac4495848
#> ── assistant [input=82 output=8 cost=$0.00] ───────
#> The total is 9.
```

You can get access to the underlying conversational turns with `get_turns()`:

```
turns <- chat$get_turns()
turns
#> [[1]]
#> <Turn: user>
#> Roll two dice and tell me the total
#>
#> [[2]]
#> <Turn: assistant>
#> [tool request (fc_0ff06e91ca3701e601690bac44710c8
#> [tool request (fc_0ff06e91ca3701e601690bac4495848
#>
#> [[3]]
#> <Turn: user>
#> [tool result  (fc_0ff06e91ca3701e601690bac44710c8
#> [tool result  (fc_0ff06e91ca3701e601690bac4495848
#>
#> [[4]]
#> <Turn: assistant>
#> The total is 9.
```

```
str(turns[[2]])
#> <ellmer::AssistantTurn>
#>  @ contents:List of 2
#>  .. $ : <ellmer::ContentToolRequest>
#>  ..  ..@ id       : chr "fc_0ff06e91ca3701e601690
#>  ..  ..@ name     : chr "tool_001"
#>  ..  ..@ arguments: Named list()
#>  ..  ..@ tool     : <ellmer::ToolDef> function ()
#>  .. .. .. @ name       : chr "tool_001"
#>  .. .. .. @ description: chr "Roll a die"
#>  .. .. .. @ arguments  : <ellmer::TypeObject>
#>  .. .. .. .. @ description          : NULL
#>  .. .. .. .. @ required             : logi TRUE
#>  .. .. .. .. @ properties           : list()
#>  .. .. .. .. @ additional_properties: logi FALSE
#>  .. .. .. @ convert    : logi TRUE
#>  .. .. .. @ annotations: list()
#>  .. $ : <ellmer::ContentToolRequest>
#>  ..  ..@ id       : chr "fc_0ff06e91ca3701e601690
#>  ..  ..@ name     : chr "tool_001"
#>  ..  ..@ arguments: Named list()
#>  ..  ..@ tool     : <ellmer::ToolDef> function ()
#>  .. .. .. @ name       : chr "tool_001"
#>  .. .. .. @ description: chr "Roll a die"
#>  .. .. .. @ arguments  : <ellmer::TypeObject>
#>  .. .. .. .. @ description          : NULL
#>  .. .. .. .. @ required             : logi TRUE
#>  .. .. .. .. @ properties           : list()
#>  .. .. .. .. @ additional_properties: logi FALSE
#>  .. .. .. @ convert    : logi TRUE
#>  .. .. .. @ annotations: list()
#>  @ text    : chr ""
#>  @ role    : chr "assistant"
#>  @ json    :List of 31
#>  .. $ id                   : chr "resp_0ff06e91c
#>  .. $ object               : chr "response"
#>  .. $ created_at           : int 1762372675
#>  .. $ status               : chr "completed"
```

```
#>  .. $ error                 : NULL
#>  .. $ incomplete_details    : NULL
#>  .. $ instructions          : NULL
#>  .. $ max_output_tokens     : NULL
#>  .. $ max_tool_calls        : NULL
#>  .. $ model                 : chr "gpt-4.1-nano-2
#>  .. $ output                :List of 2
#>  ..  ..$ :List of 6
#>  ..  .. ..$ id       : chr "fc_0ff06e91ca3701e601
#>  ..  .. ..$ type     : chr "function_call"
#>  ..  .. ..$ status   : chr "completed"
#>  ..  .. ..$ arguments: chr "{}"
#>  ..  .. ..$ call_id  : chr "call_UoyOnszXDnPApY7C
#>  ..  .. ..$ name     : chr "tool_001"
#>  ..  ..$ :List of 6
#>  ..  .. ..$ id       : chr "fc_0ff06e91ca3701e601
#>  ..  .. ..$ type     : chr "function_call"
#>  ..  .. ..$ status   : chr "completed"
#>  ..  .. ..$ arguments: chr "{}"
#>  ..  .. ..$ call_id  : chr "call_6k0VAH5JjUWtIh4c
#>  ..  .. ..$ name     : chr "tool_001"
#>  .. $ parallel_tool_calls   : logi TRUE
#>  .. $ previous_response_id  : NULL
#>  .. $ prompt_cache_key      : NULL
#>  .. $ prompt_cache_retention: NULL
#>  .. $ reasoning             :List of 2
#>  ..  ..$ effort : NULL
#>  ..  ..$ summary: NULL
#>  .. $ safety_identifier     : NULL
#>  .. $ service_tier          : chr "default"
#>  .. $ store                 : logi FALSE
#>  .. $ temperature           : num 1
#>  .. $ text                  :List of 2
#>  ..  ..$ format   :List of 1
#>  ..  .. ..$ type: chr "text"
#>  ..  ..$ verbosity: chr "medium"
#>  .. $ tool_choice           : chr "auto"
#>  .. $ tools                 :List of 1
#>  ..  ..$ :List of 5
#>  ..  .. ..$ type       : chr "function"
```

```
#>  ..  .. .. ..$ type                 : chr "object"
#>  ..  .. .. ..$ description          : chr ""
#>  ..  .. .. ..$ properties           : Named list()
#>  ..  .. .. ..$ required             : list()
#>  ..  .. .. ..$ additionalProperties: logi FALSE
#>  ..  .. ..$ strict     : logi TRUE
#>  .. $ top_logprobs           : int 0
#>  .. $ top_p                  : num 1
#>  .. $ truncation             : chr "disabled"
#>  .. $ usage                  :List of 5
#>  ..  ..$ input_tokens          : int 22
#>  ..  ..$ input_tokens_details :List of 1
#>  ..  .. ..$ cached_tokens: int 0
#>  ..  ..$ output_tokens         : int 42
#>  ..  ..$ output_tokens_details:List of 1
#>  ..  .. ..$ reasoning_tokens: int 0
#>  ..  ..$ total_tokens          : int 64
#>  .. $ user                   : NULL
#>  .. $ metadata               : Named list()
#>  @ tokens  : Named int [1:3] 22 42 0
#>  .. - attr(*, "names")= chr [1:3] "input" "output"
#>  @ cost    : 'ellmer_dollars' num $0.00
#>  @ duration: num NA
```

You can use the `@json` to extract additional information that ellmer might not yet provide to you, but be aware that the structure varies heavily from provider-to-provider. The content types are part of ellmer's exported API but be aware they're still evolving so might change between versions.

Developed by Hadley Wickham, Joe Cheng, Aaron Jacobs, Garrick Aden-Buie, Barret Schloerke, posit. Site built with pkgdown 2.2.0.