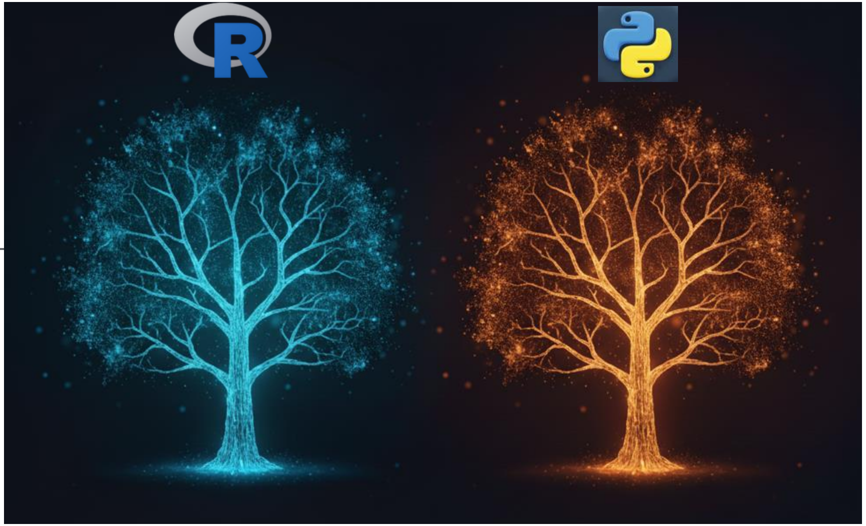
# Tracking the Evolution of R and Python Tools for GenAI:

## State of the Ecosystem & Lessons from Building in Both



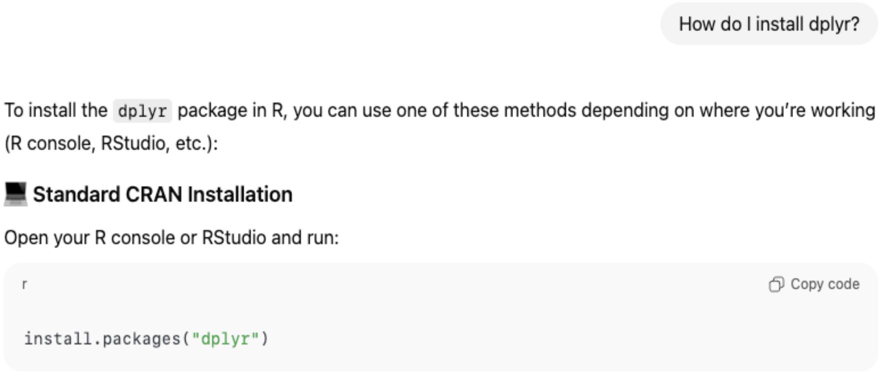
Xu Fei
Senior Solutions Engineer
xu@a2-ai.com
linkedin.com/in/xufei1/
xuf12

# Who is this talk for

- Language: R-primary or multilingual teams
- Background: Have experience building apps, interested in adding GenAI features (e.g. chatbot, coding assistant)

# Our experience – chatbot for internal tools

ChatGPT
(generic chatbot)



How do I install dplyr?

To install the `dplyr` package in R, you can use one of these methods depending on where you're working (R console, RStudio, etc.):

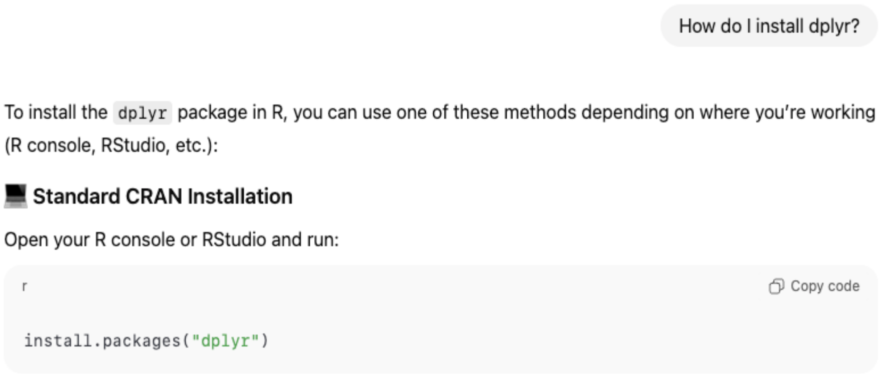
💻 **Standard CRAN Installation**

Open your R console or RStudio and run:

```r
install.packages("dplyr")
```

# Our experience – chatbot for internal tools

## ChatGPT
## (generic chatbot)



How do I install dplyr?

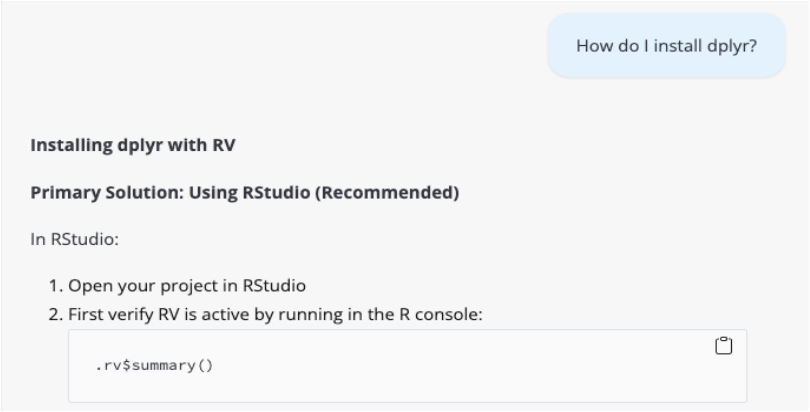
To install the `dplyr` package in R, you can use one of these methods depending on where you're working (R console, RStudio, etc.):

**💻 Standard CRAN Installation**

Open your R console or RStudio and run:

```r
install.packages("dplyr")
```

## PharmaSage
## (uses internal docs + code)



How do I install dplyr?

**Installing dplyr with RV**

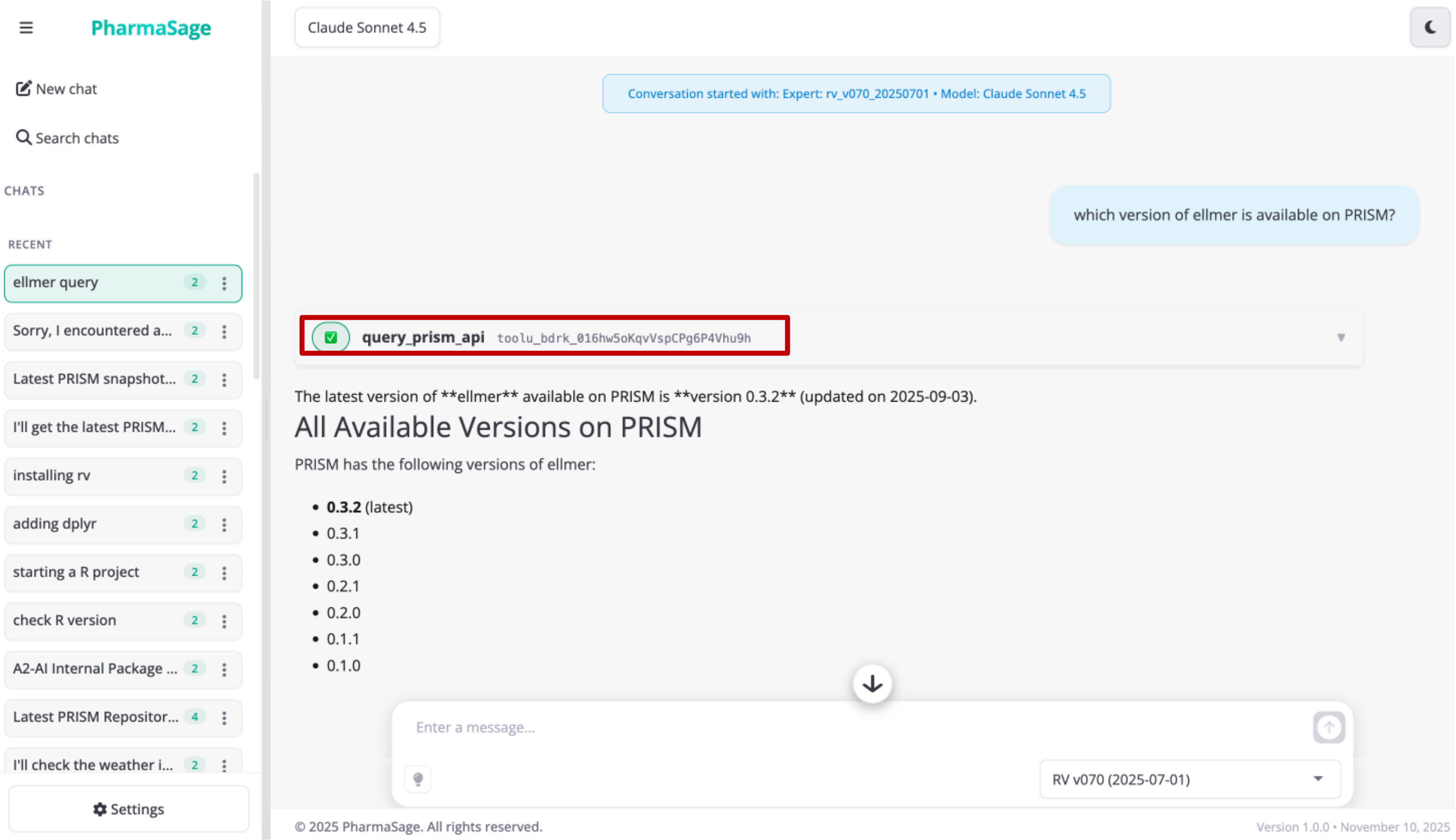
**Primary Solution: Using RStudio (Recommended)**

In RStudio:

1. Open your project in RStudio
2. First verify RV is active by running in the R console:

```
.rv$summary()
```

# Our experience – chatbot with tool calling



Conversation started with: Expert: rv_v070_20250701 • Model: Claude Sonnet 4.5

which version of ellmer is available on PRISM?

☑ **query_prism_api** `toolu_bdrk_016hw5oKqvVspCPg6P4Vhu9h`

The latest version of **ellmer** available on PRISM is **version 0.3.2** (updated on 2025-09-03).

## All Available Versions on PRISM

PRISM has the following versions of ellmer:

- **0.3.2** (latest)
- 0.3.1
- 0.3.0
- 0.2.1
- 0.2.0
- 0.1.1
- 0.1.0

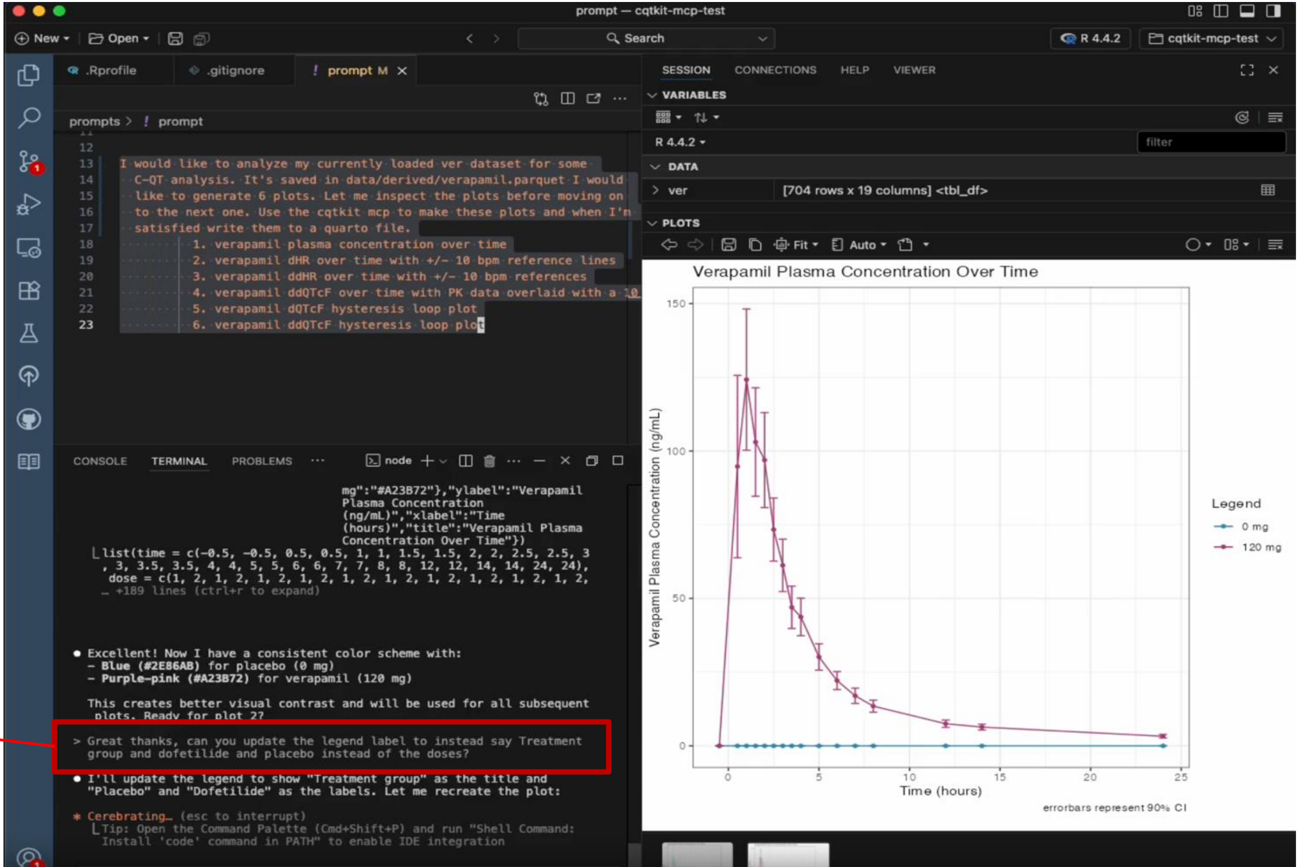
© 2025 PharmaSage. All rights reserved.

Version 1.0.0 • November 10, 2025

# Our experience – MCP server built in R

## Enables natural language queries to do exploratory data analysis

> Great thanks, can you update the legend label to instead say Treatment group and dofetilide and placebo instead of the doses?



```
I would like to analyze my currently loaded ver dataset for some
 C-QT analysis. It's saved in data/derived/verapamil.parquet I would
 like to generate 6 plots. Let me inspect the plots before moving on
 to the next one. Use the cqtkit mcp to make these plots and when I'm
 satisfied write them to a quarto file.
        1. verapamil plasma concentration over time
        2. verapamil dHR over time with +/- 10 bpm reference lines
        3. verapamil ddHR over time with +/- 10 bpm references
        4. verapamil ddQTcF over time with PK data overlaid with a 10
        5. verapamil dQTcF hysteresis loop plot
        6. verapamil ddQTcF hysteresis loop plot
```

```
                                   mg":"#A23B72"},"ylabel":"Verapamil
                                   Plasma Concentration
                                   (ng/mL)","xlabel":"Time
                                   (hours)","title":"Verapamil Plasma
                                   Concentration Over Time"})
  ⎿ list(time = c(-0.5, -0.5, 0.5, 0.5, 1, 1, 1.5, 1.5, 2, 2, 2.5, 2.5, 3
    , 3, 3.5, 3.5, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 12, 12, 14, 14, 24, 24),
     dose = c(1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
    … +189 lines (ctrl+r to expand)


● Excellent! Now I have a consistent color scheme with:
  - Blue (#2E86AB) for placebo (0 mg)
  - Purple-pink (#A23B72) for verapamil (120 mg)

  This creates better visual contrast and will be used for all subsequent
   plots. Ready for plot 2?

> Great thanks, can you update the legend label to instead say Treatment
  group and dofetilide and placebo instead of the doses?

● I'll update the legend to show "Treatment group" as the title and
  "Placebo" and "Dofetilide" as the labels. Let me recreate the plot:

✱ Cerebrating… (esc to interrupt)
  ⎿ Tip: Open the Command Palette (Cmd+Shift+P) and run "Shell Command:
    Install 'code' command in PATH" to enable IDE integration
```

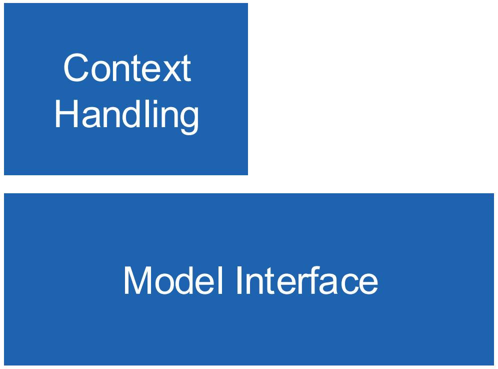
# 3 core building blocks of GenAI applications

1. Model Interface
    - Managing LLM API calls

Model Interface

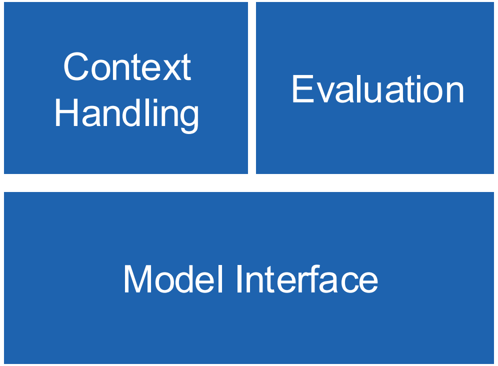
# 3 core building blocks of GenAI applications

1. Model Interface
    - Managing LLM API calls
2. Context Handling
    - Handling anything that goes into the LLM



# 3 core building blocks of GenAI applications

1. Model Interface
   - Managing LLM API calls
2. Context Handling
   - Handling anything that goes into the LLM
3. Evaluation
   - Validating the results



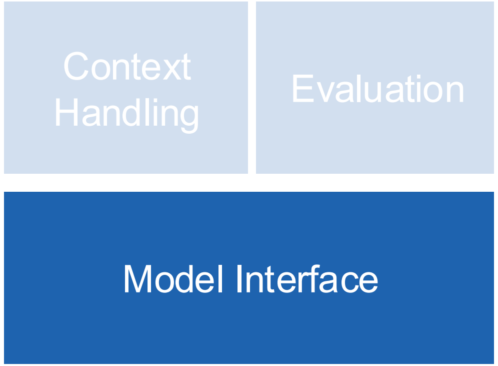
Context Handling

Evaluation

Model Interface

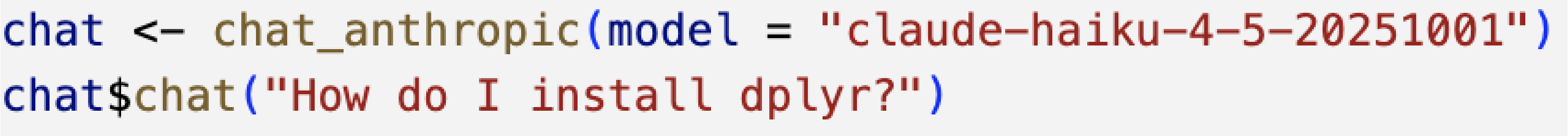
# 3 core building blocks of GenAI applications

1. Model Interface
   - Managing LLM API calls
2. Context Handling
   - Handling anything that goes into the LLM
3. Evaluation
   - Validating the results



# Model Interface

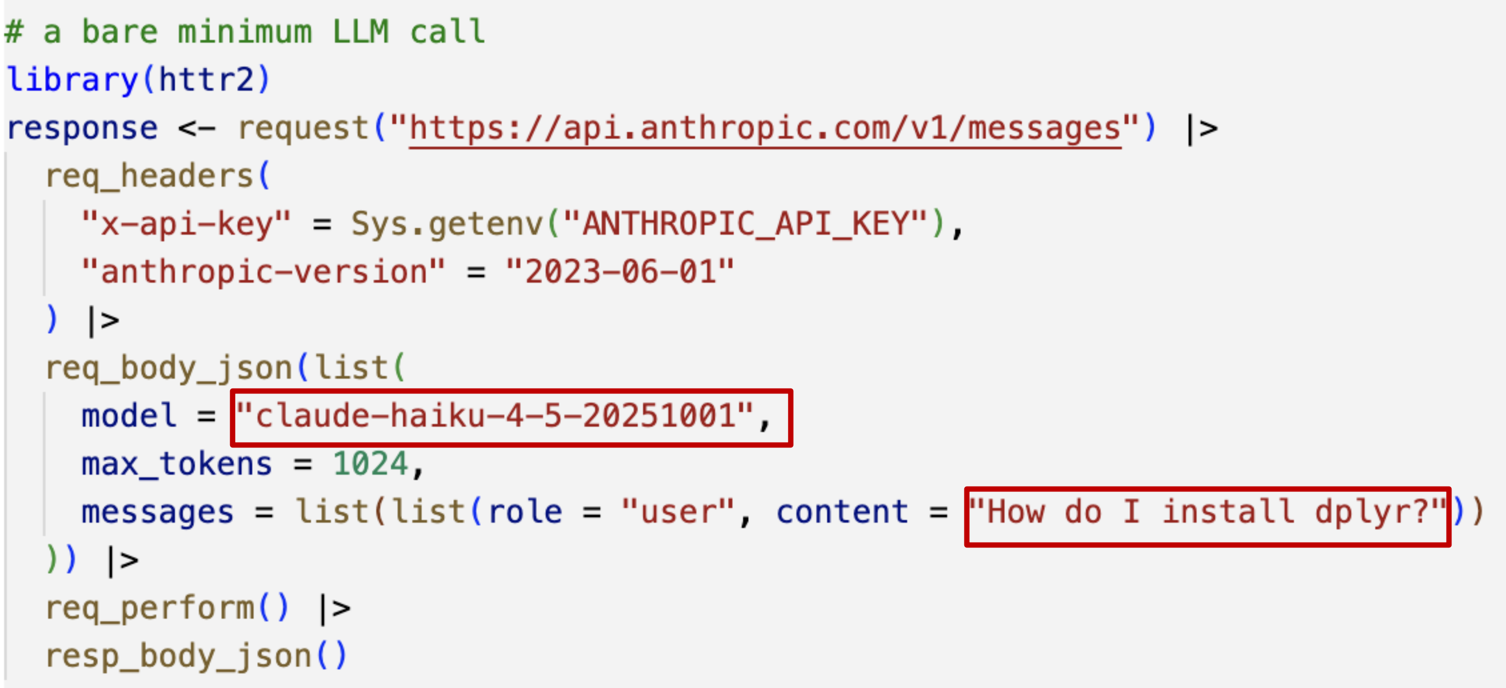
## You might like to ask an LLM a question in this way…



```r
chat <- chat_anthropic(model = "claude-haiku-4-5-20251001")
chat$chat("How do I install dplyr?")
```

# Model Interface

…but sending this question in raw HTTP actually looks like this



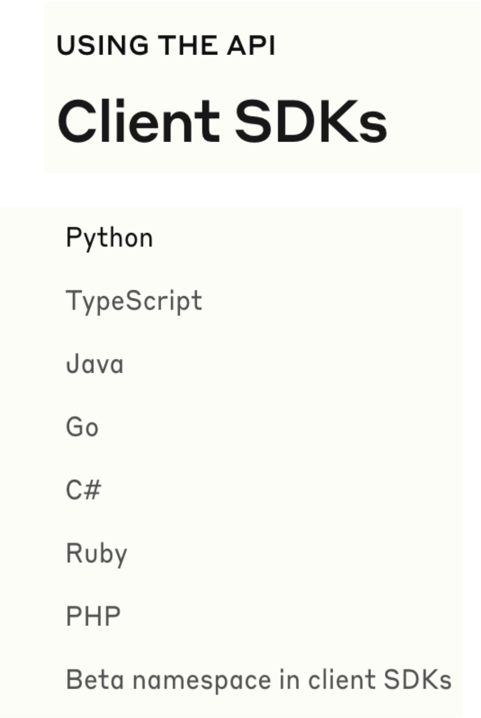
```r
# a bare minimum LLM call
library(httr2)
response <- request("https://api.anthropic.com/v1/messages") |>
  req_headers(
    "x-api-key" = Sys.getenv("ANTHROPIC_API_KEY"),
    "anthropic-version" = "2023-06-01"
  ) |>
  req_body_json(list(
    model = "claude-haiku-4-5-20251001",
    max_tokens = 1024,
    messages = list(list(role = "user", content = "How do I install dplyr?"))
  )) |>
  req_perform() |>
  resp_body_json()
```

# Model Interface – client SDKs

http calls
client SDKs
imgflip.com

# Model Interface – SDK

## Anthropic



USING THE API

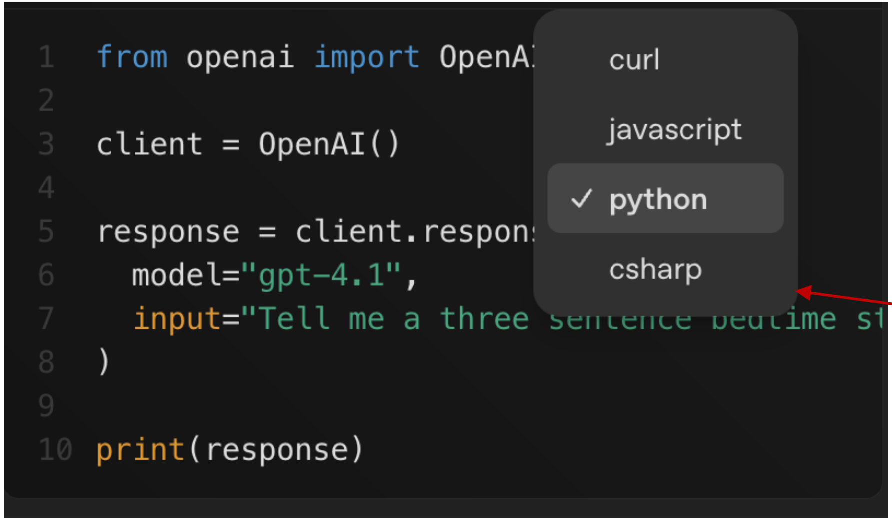
**Client SDKs**

- Python
- TypeScript
- Java
- Go
- C#
- Ruby
- PHP
- Beta namespace in client SDKs

Where is R?

# Model Interface – SDK

OpenAI



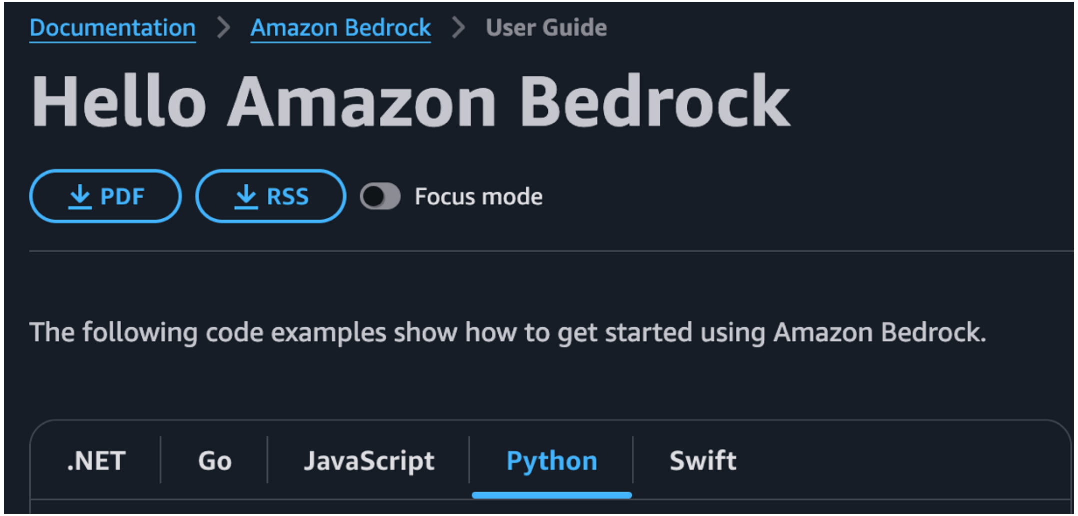
```
from openai import OpenA

client = OpenAI()

response = client.respon
  model="gpt-4.1",
  input="Tell me a three sentence bedtime st
)

print(response)
```

Also no R…

https://platform.openai.com/docs/api-reference/responses/create?lang=python

# Model Interface – SDK

## Amazon Bedrock



Documentation > Amazon Bedrock > User Guide

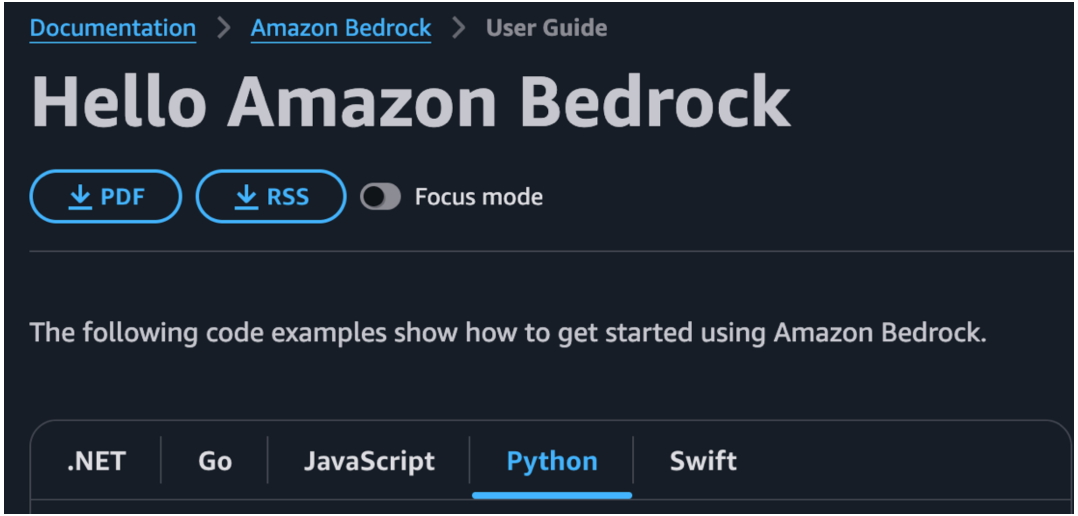
Hello Amazon Bedrock

The following code examples show how to get started using Amazon Bedrock.

Still no R ☹

# Model Interface – SDK

Amazon Bedrock



Documentation > Amazon Bedrock > User Guide

Hello Amazon Bedrock

The following code examples show how to get started using Amazon Bedrock.

but

# Model Interface – pick a direction?

https://www.istockphoto.com/photo/business-woman-in-front-of-two-roads-thinking-deciding-gm564598634-99029991

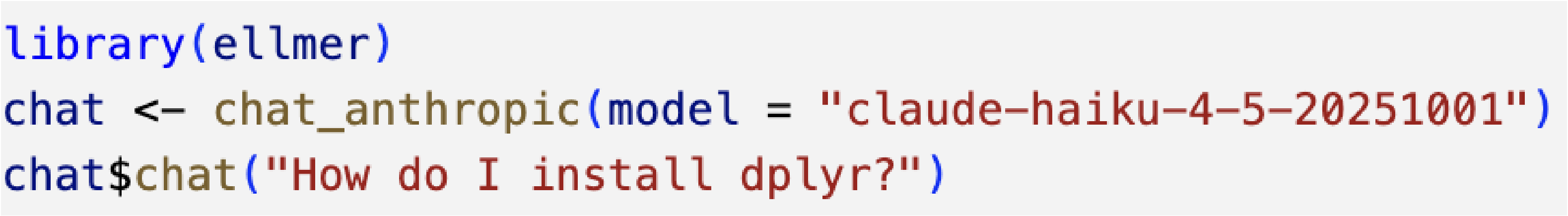
# Model Interface – the answer for R

Supports Anthropic, OpenAI, AWS Bedrock, Gemini, etc.

0.3.2 (latest) released on 2025-09-03
0.1.0 released on 2025-01-09

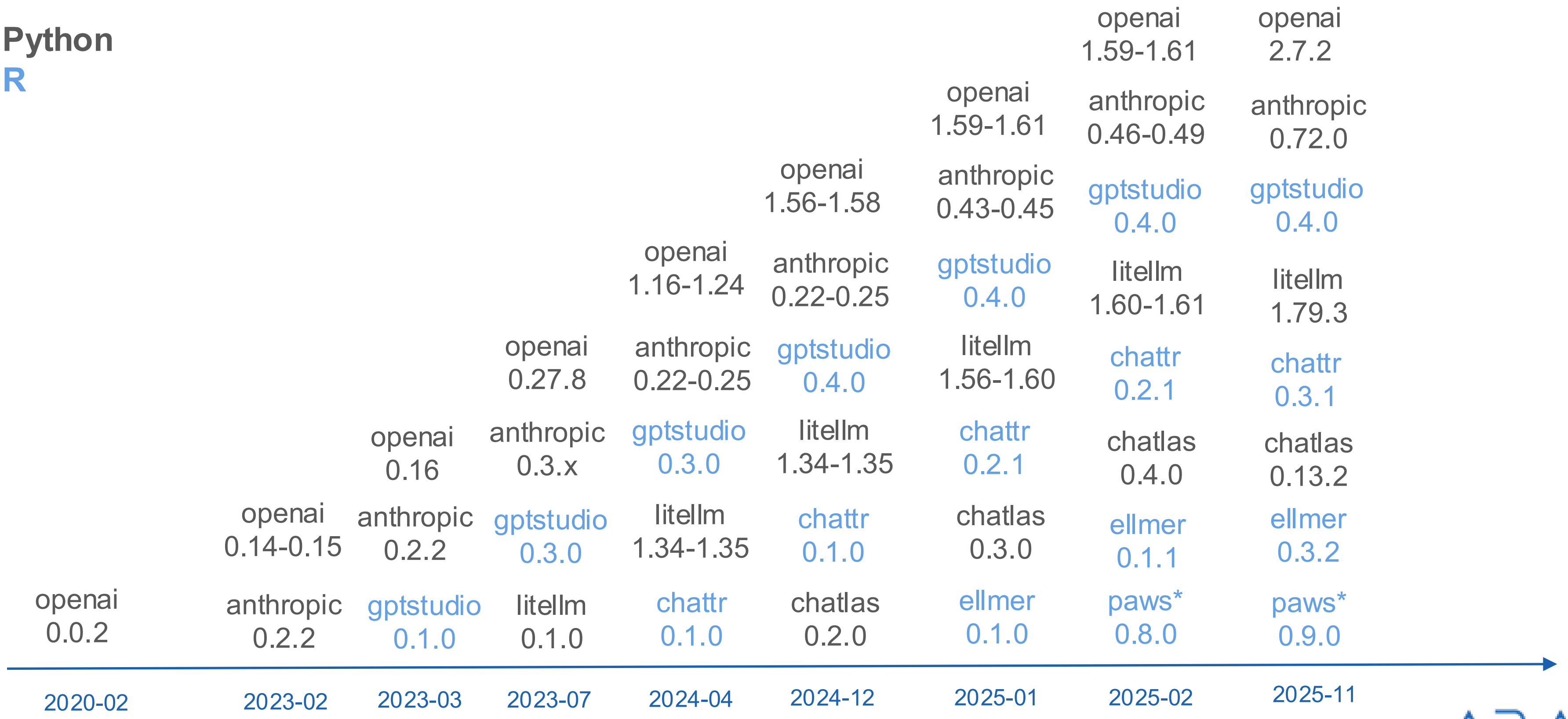
https://ellmer.tidyverse.org/index.html

# Model Interface – ellmer



```r
library(ellmer)
chat <- chat_anthropic(model = "claude-haiku-4-5-20251001")
chat$chat("How do I install dplyr?")
```

# Model Interface – ecosystem evolution



**Python**
**R**

openai
0.0.2

2020-02

openai
0.14-0.15
anthropic
0.2.2

2023-02

openai
0.16
anthropic
0.2.2
gptstudio
0.1.0

2023-03

openai
0.27.8
anthropic
0.3.x
gptstudio
0.3.0
litellm
0.1.0

2023-07

openai
1.16-1.24
anthropic
0.22-0.25
gptstudio
0.3.0
litellm
1.34-1.35
chattr
0.1.0

2024-04

openai
1.56-1.58
anthropic
0.22-0.25
gptstudio
0.4.0
litellm
1.34-1.35
chattr
0.1.0
chatlas
0.2.0

2024-12

openai
1.59-1.61
anthropic
0.43-0.45
gptstudio
0.4.0
litellm
1.56-1.60
chattr
0.2.1
chatlas
0.3.0
ellmer
0.1.0

2025-01

openai
1.59-1.61
anthropic
0.46-0.49
gptstudio
0.4.0
litellm
1.60-1.61
chattr
0.2.1
chatlas
0.4.0
ellmer
0.1.1
paws*
0.8.0

2025-02

openai
2.7.2
anthropic
0.72.0
gptstudio
0.4.0
litellm
1.79.3
chattr
0.3.1
chatlas
0.13.2
ellmer
0.3.2
paws*
0.9.0

2025-11

* Not only for LLM model interface

# Model Interface – abstraction level

| | R | Python |
|---|---|---|
| High-level orchestration frameworks | ? | langchain<br>llamaindex<br>autogen<br>etc. |
| Unified LLM abstraction | ellmer<br>gptstudio<br>chattr | litellm<br>chatlas<br>etc. |
| LLM provider SDKs | paws* | openai<br>anthropic<br>gemini<br>etc. |

LLM provider REST API

---

* Not only for LLM model interface

# Model Interface – # of monthly downloads

**Python**
**R**

| | **monthly downloads across all versions** |
|---|---|
| openai | 128M |
| litellm | 39M |
| anthropic | 29M |
| chatlas | 16.5k |
| ellmer | 9.2k |
| paws* | 4.1k |
| chattr | 1.5k |
| gptstudio | 1.1k |

Numbers calculated using R library “cranlogs” and pypistats.org as of November 10, 2025

* Not only for LLM model interface

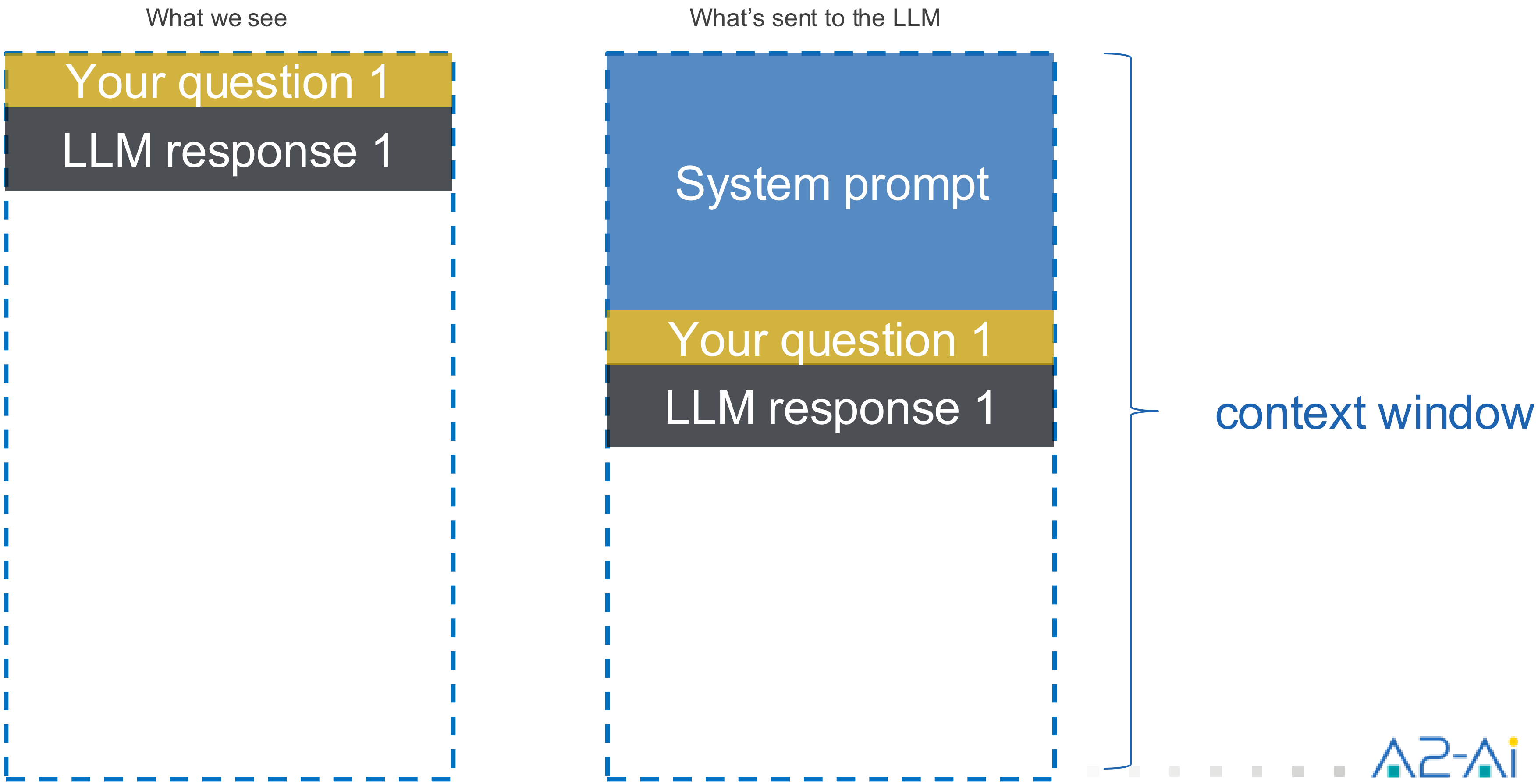
# Model Interface – case study

## Able to extend model interface using Python SDK

### Need:

We need prompt caching to reduce costs in AWS Bedrock.

# Context handling: what’s in a context window?



What we see

Your question 1

LLM response 1

What’s sent to the LLM

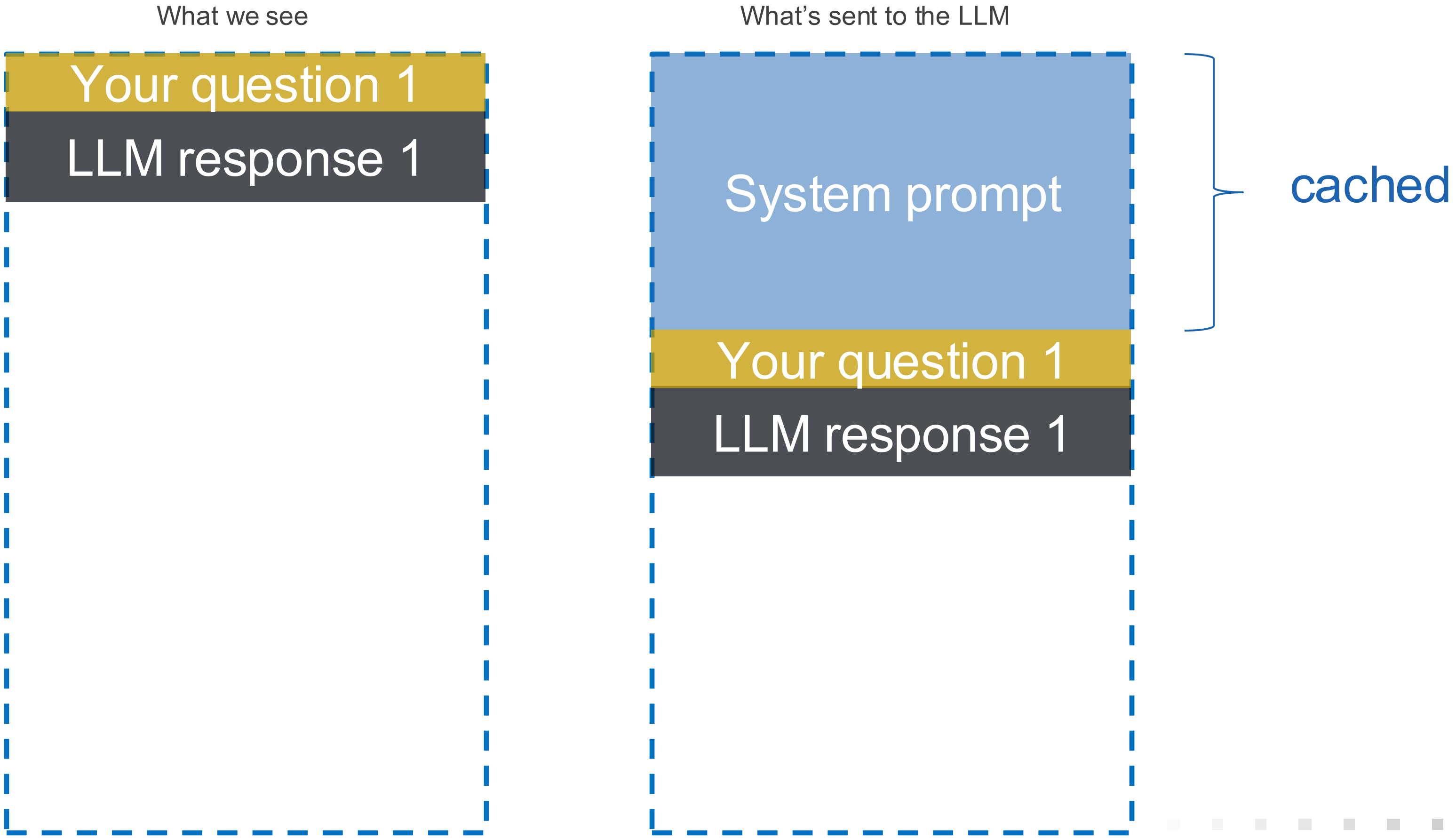
System prompt

Your question 1

LLM response 1

context window

# Context handling: what's cached?



What we see

Your question 1

LLM response 1

What's sent to the LLM

System prompt

Your question 1

LLM response 1

cached

# Model Interface – case study

## Able to extend model interface using Python SDK

### Need:

We need prompt caching to reduce costs in AWS Bedrock.

### Challenge:

In September 2025, it was not part of the chatlas package for the Anthropic models that we used.

# Model Interface – case study

## Able to extend model interface using Python SDK

### Need:

We need prompt caching to reduce costs in AWS Bedrock.

### Challenge:

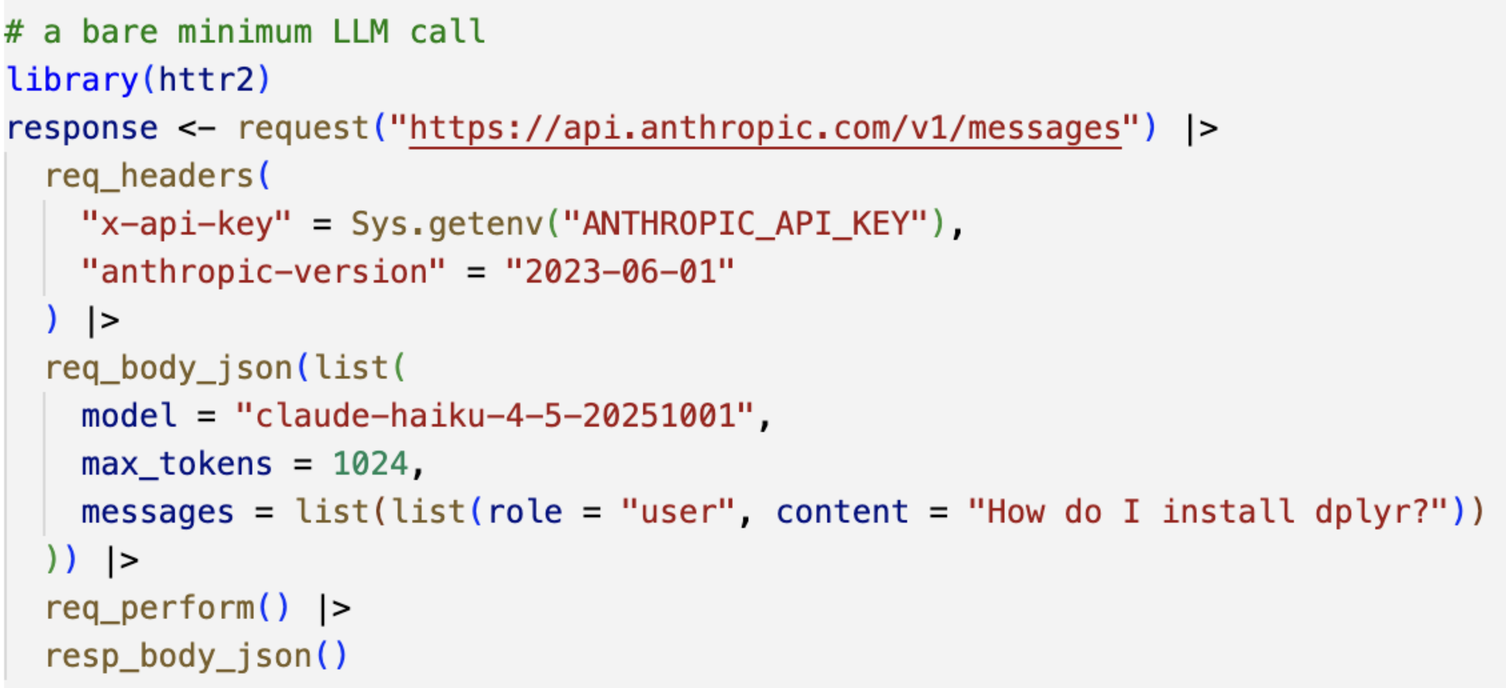
In September 2025, it was not part of the chatlas package for the Anthropic models that we used.

### Solution:

We decided to add it to chatlas ourselves, because prompt caching was in the AWS Bedrock Python SDK.

# Model Interface – REST API

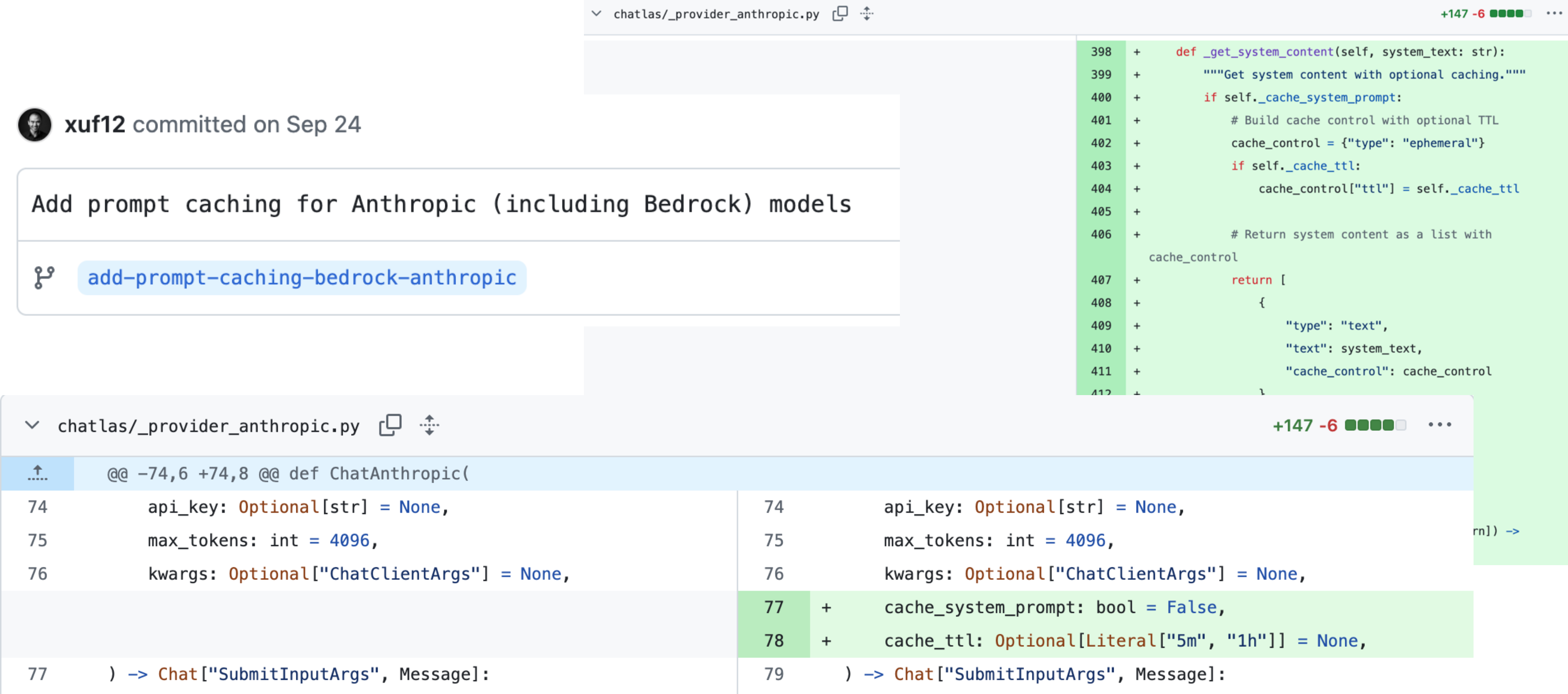
## Without the SDK, we would have to deal with raw HTTP directly



```
# a bare minimum LLM call
library(httr2)
response <- request("https://api.anthropic.com/v1/messages") |>
  req_headers(
    "x-api-key" = Sys.getenv("ANTHROPIC_API_KEY"),
    "anthropic-version" = "2023-06-01"
  ) |>
  req_body_json(list(
    model = "claude-haiku-4-5-20251001",
    max_tokens = 1024,
    messages = list(list(role = "user", content = "How do I install dplyr?"))
  )) |>
  req_perform() |>
  resp_body_json()
```

# Model Interface – adding a new feature

## Able to extend model interface using Python SDK



```
+     def _get_system_content(self, system_text: str):
+         """Get system content with optional caching."""
+         if self._cache_system_prompt:
+             # Build cache control with optional TTL
+             cache_control = {"type": "ephemeral"}
+             if self._cache_ttl:
+                 cache_control["ttl"] = self._cache_ttl
+
+             # Return system content as a list with
   cache_control
+             return [
+                 {
+                     "type": "text",
+                     "text": system_text,
+                     "cache_control": cache_control
rn]) ->
```

**xuf12** committed on Sep 24

Add prompt caching for Anthropic (including Bedrock) models

add-prompt-caching-bedrock-anthropic

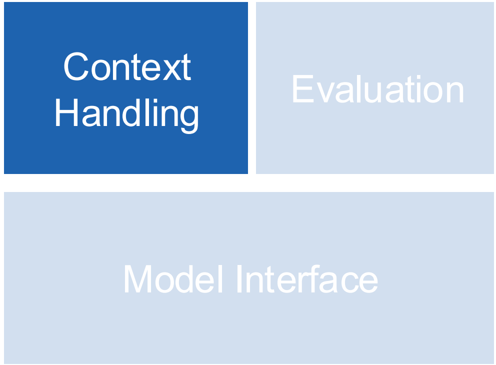
```
@@ -74,6 +74,8 @@ def ChatAnthropic(
```

```
        api_key: Optional[str] = None,
        max_tokens: int = 4096,
        kwargs: Optional["ChatClientArgs"] = None,
    ) -> Chat["SubmitInputArgs", Message]:
```

```
        api_key: Optional[str] = None,
        max_tokens: int = 4096,
        kwargs: Optional["ChatClientArgs"] = None,
+       cache_system_prompt: bool = False,
+       cache_ttl: Optional[Literal["5m", "1h"]] = None,
    ) -> Chat["SubmitInputArgs", Message]:
```

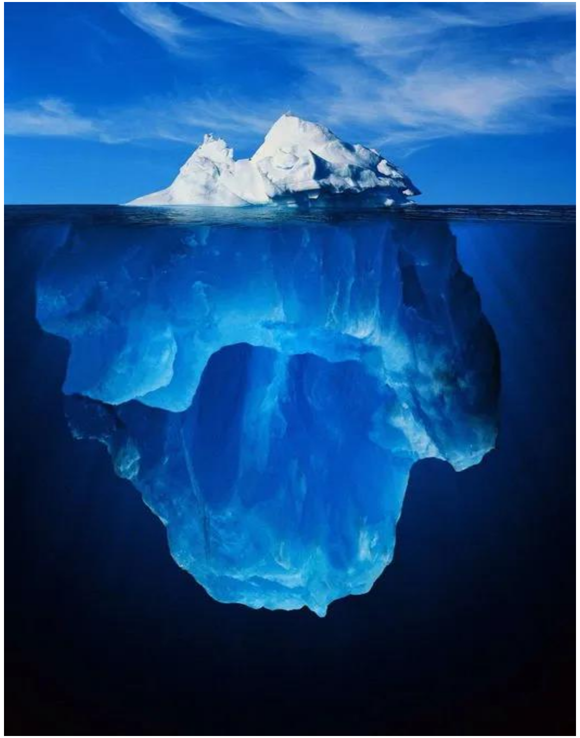
# 3 core building blocks of GenAI applications

1. Model Interface
   - Managing LLM API calls
2. Context Handling
   - Handling anything that goes into the LLM
3. Evaluation
   - Validating the results are as intended



# Context handling

## Example for a knowledge assistant app



← Submitting questions via model interface

all of the work to process enterprise-specific documents and code to share with the model

(requires additional tooling)

http://mailchi.mp/762ea4c14631/are-you-seeing-only-the-tip-of-the-iceberg-move-to-learn-therapy-news?e=d5fdf2d9a8

# Context handling – pick a direction again?

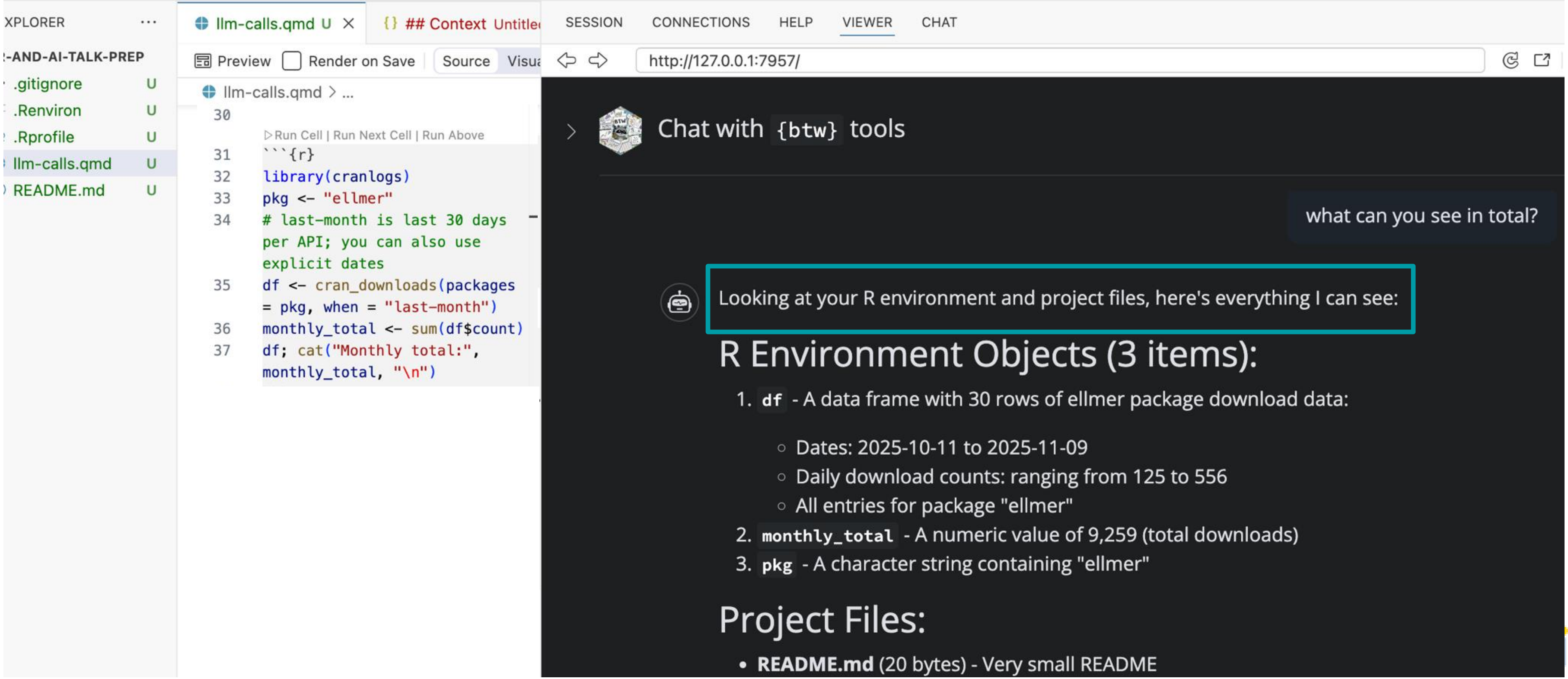
https://www.istockphoto.com/photo/business-woman-in-front-of-two-roads-thinking-deciding-gm564598634-99029991

# Context handling

1. In R, start with ellmer and btw

# Context handling

## btw can use your R session as context



~~~
```{r}
library(cranlogs)
pkg <- "ellmer"
# last-month is last 30 days per API; you can also use explicit dates
df <- cran_downloads(packages = pkg, when = "last-month")
monthly_total <- sum(df$count)
df; cat("Monthly total:", monthly_total, "\n")
~~~

Chat with `{btw}` tools

what can you see in total?

Looking at your R environment and project files, here's everything I can see:

# R Environment Objects (3 items):

1. `df` - A data frame with 30 rows of ellmer package download data:
   - Dates: 2025-10-11 to 2025-11-09
   - Daily download counts: ranging from 125 to 556
   - All entries for package "ellmer"
2. `monthly_total` - A numeric value of 9,259 (total downloads)
3. `pkg` - A character string containing "ellmer"

# Project Files:

- **README.md** (20 bytes) - Very small README

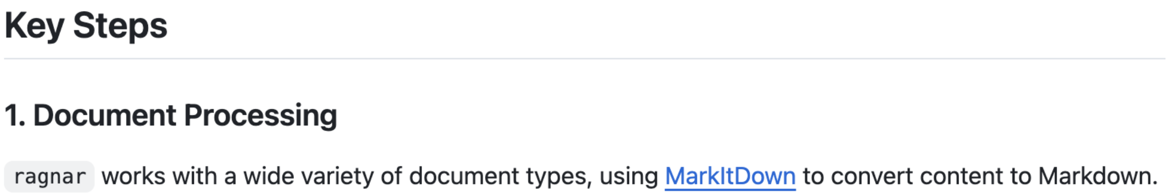
# Context handling

## Options when required libraries don’t exist in R

1. In R, start with ellmer and btw
2. When required libraries don't exist in R, try to use reticulate to call specific Python libraries and functions

# Context handling

## Options when required libraries don’t exist in R

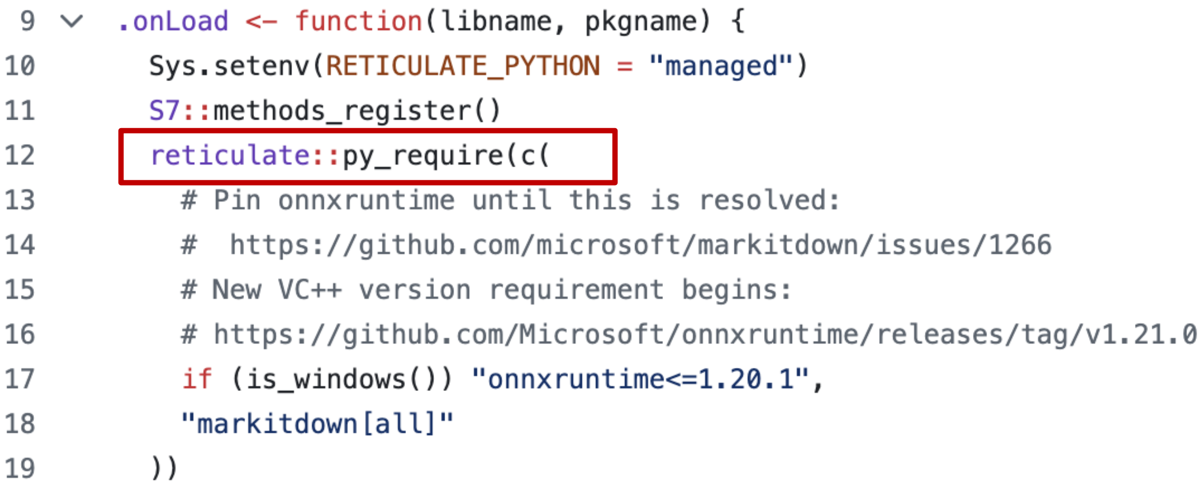


### Key Steps

#### 1. Document Processing

`ragnar` works with a wide variety of document types, using MarkItDown to convert content to Markdown.

**ragnar / R / ragnar-package.R**



```r
.onLoad <- function(libname, pkgname) {
  Sys.setenv(RETICULATE_PYTHON = "managed")
  S7::methods_register()
  reticulate::py_require(c(
    # Pin onnxruntime until this is resolved:
    #  https://github.com/microsoft/markitdown/issues/1266
    # New VC++ version requirement begins:
    # https://github.com/Microsoft/onnxruntime/releases/tag/v1.21.0
    if (is_windows()) "onnxruntime<=1.20.1",
    "markitdown[all]"
  ))
```

# Context handling

## For multilingual teams

1. In R, start with ellmer and btw

2. When required libraries don't exist in R, try to use reticulate to call specific Python libraries and functions

3*. Create REST APIs for existing R (plumber) or Python (FastAPI) tools

# Context handling

## For multilingual teams

1. In R, start with ellmer and btw

2. When required libraries don't exist in R, try to use reticulate to call specific Python libraries and functions

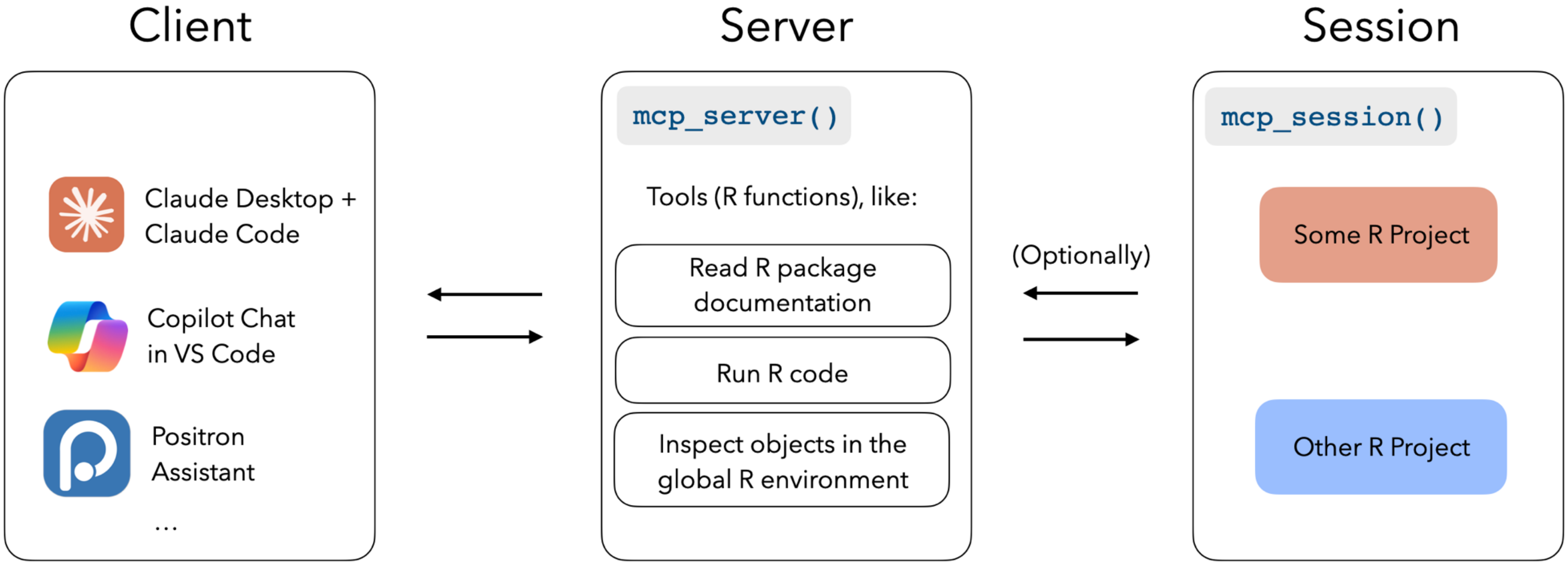
3*. Create REST APIs for existing R (plumber) or Python (FastAPI) tools

4*. Build a Model Context Protocol (MCP) server

**Model Context Protocol**

# Context handling

## MCP enables language-agnostic calls to use functions



**Client**

Claude Desktop + Claude Code

Copilot Chat in VS Code

Positron Assistant

...

**Server**

`mcp_server()`

Tools (R functions), like:

- Read R package documentation
- Run R code
- Inspect objects in the global R environment

(Optionally)

**Session**

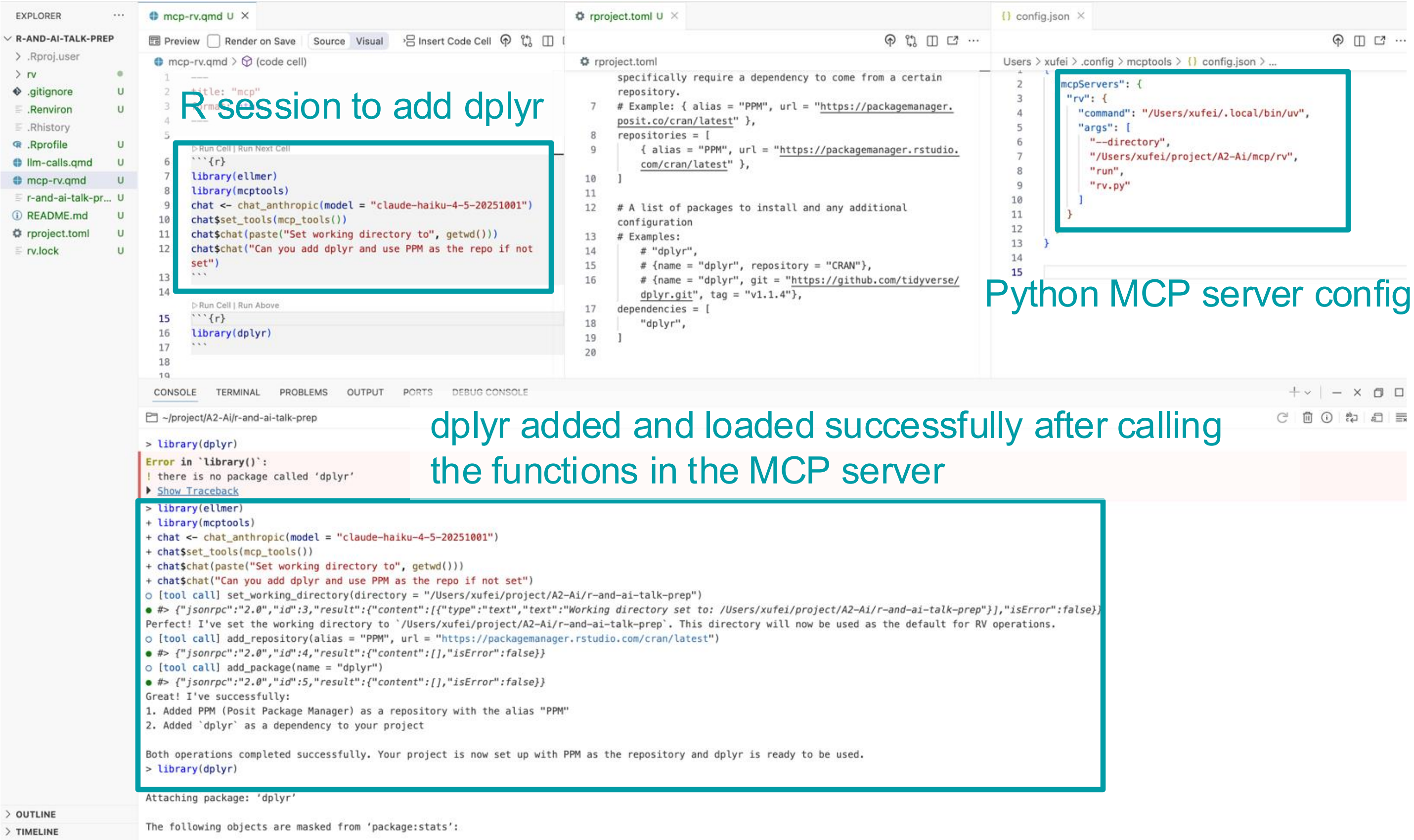
`mcp_session()`

Some R Project

Other R Project

# Context handling

## Calling a Python MCP server in an R session to install an R package



R session to add dplyr

~~~
---
title: "mcp"
format: [illegible]
---

```{r}
library(ellmer)
library(mcptools)
chat <- chat_anthropic(model = "claude-haiku-4-5-20251001")
chat$set_tools(mcp_tools())
chat$chat(paste("Set working directory to", getwd()))
chat$chat("Can you add dplyr and use PPM as the repo if not set")
```

```{r}
library(dplyr)
```
~~~

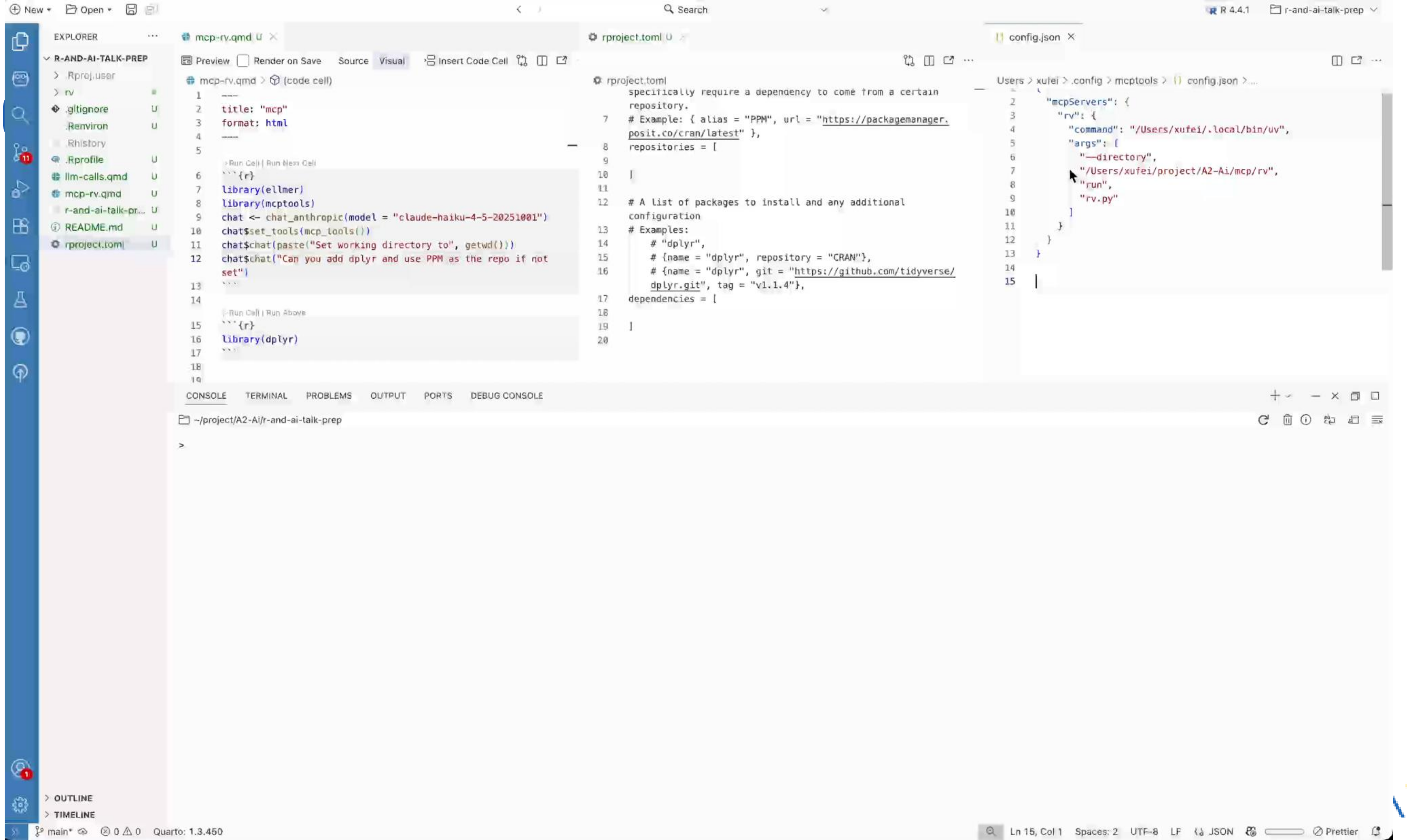
```
specifically require a dependency to come from a certain repository.
# Example: { alias = "PPM", url = "https://packagemanager.posit.co/cran/latest" },
repositories = [
    { alias = "PPM", url = "https://packagemanager.rstudio.com/cran/latest" },
]

# A list of packages to install and any additional configuration
# Examples:
    # "dplyr",
    # {name = "dplyr", repository = "CRAN"},
    # {name = "dplyr", git = "https://github.com/tidyverse/dplyr.git", tag = "v1.1.4"},
dependencies = [
    "dplyr",
]
```

Python MCP server config file

```
  mcpServers": {
   "rv": {
     "command": "/Users/xufei/.local/bin/uv",
     "args": [
       "--directory",
       "/Users/xufei/project/A2-Ai/mcp/rv",
       "run",
       "rv.py"
     ]
   }
 }
}
```

dplyr added and loaded successfully after calling the functions in the MCP server

```
> library(dplyr)
Error in `library()`:
! there is no package called ‘dplyr’
▸ Show Traceback
> library(ellmer)
+ library(mcptools)
+ chat <- chat_anthropic(model = "claude-haiku-4-5-20251001")
+ chat$set_tools(mcp_tools())
+ chat$chat(paste("Set working directory to", getwd()))
+ chat$chat("Can you add dplyr and use PPM as the repo if not set")
○ [tool call] set_working_directory(directory = "/Users/xufei/project/A2-Ai/r-and-ai-talk-prep")
● #> {"jsonrpc":"2.0","id":3,"result":{"content":[{"type":"text","text":"Working directory set to: /Users/xufei/project/A2-Ai/r-and-ai-talk-prep"}],"isError":false}}
Perfect! I've set the working directory to `/Users/xufei/project/A2-Ai/r-and-ai-talk-prep`. This directory will now be used as the default for RV operations.
○ [tool call] add_repository(alias = "PPM", url = "https://packagemanager.rstudio.com/cran/latest")
● #> {"jsonrpc":"2.0","id":4,"result":{"content":[],"isError":false}}
○ [tool call] add_package(name = "dplyr")
● #> {"jsonrpc":"2.0","id":5,"result":{"content":[],"isError":false}}
Great! I've successfully:
1. Added PPM (Posit Package Manager) as a repository with the alias "PPM"
2. Added `dplyr` as a dependency to your project

Both operations completed successfully. Your project is now set up with PPM as the repository and dplyr is ready to be used.
> library(dplyr)

Attaching package: ‘dplyr’

The following objects are masked from ‘package:stats’:
```



~~~
---
title: "mcp"
format: html
---

```{r}
library(ellmer)
library(mcptools)
chat <- chat_anthropic(model = "claude-haiku-4-5-20251001")
chat$set_tools(mcp_tools())
chat$chat(paste("Set working directory to", getwd()))
chat$chat("Can you add dplyr and use PPM as the repo if not
set")
```

```{r}
library(dplyr)
```
~~~

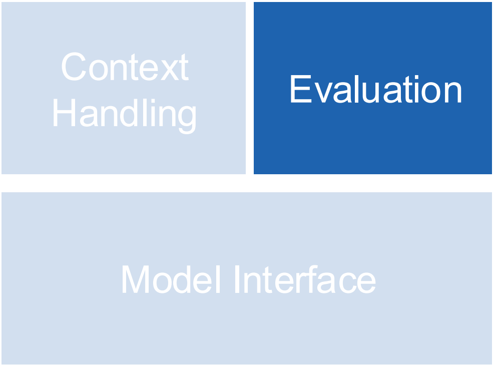
```
specifically require a dependency to come from a certain
repository.
# Example: { alias = "PPM", url = "https://packagemanager.
posit.co/cran/latest" },
repositories = [

]

# A list of packages to install and any additional
configuration
# Examples:
    # "dplyr",
    # {name = "dplyr", repository = "CRAN"},
    # {name = "dplyr", git = "https://github.com/tidyverse/
    dplyr.git", tag = "v1.1.4"},
dependencies = [

]
```

```
  "mcpServers": {
    "rv": {
      "command": "/Users/xufei/.local/bin/uv",
      "args": [
        "--directory",
        "/Users/xufei/project/A2-Ai/mcp/rv",
        "run",
        "rv.py"
      ]
    }
  }
}
```

```
~/project/A2-Ai/r-and-ai-talk-prep

>
```

# 3 core building blocks of GenAI applications

1. Model Interface
   - Managing LLM API calls
2. Context Handling
   - Handling anything that goes into the LLM
3. Evaluation
   - Validating the results

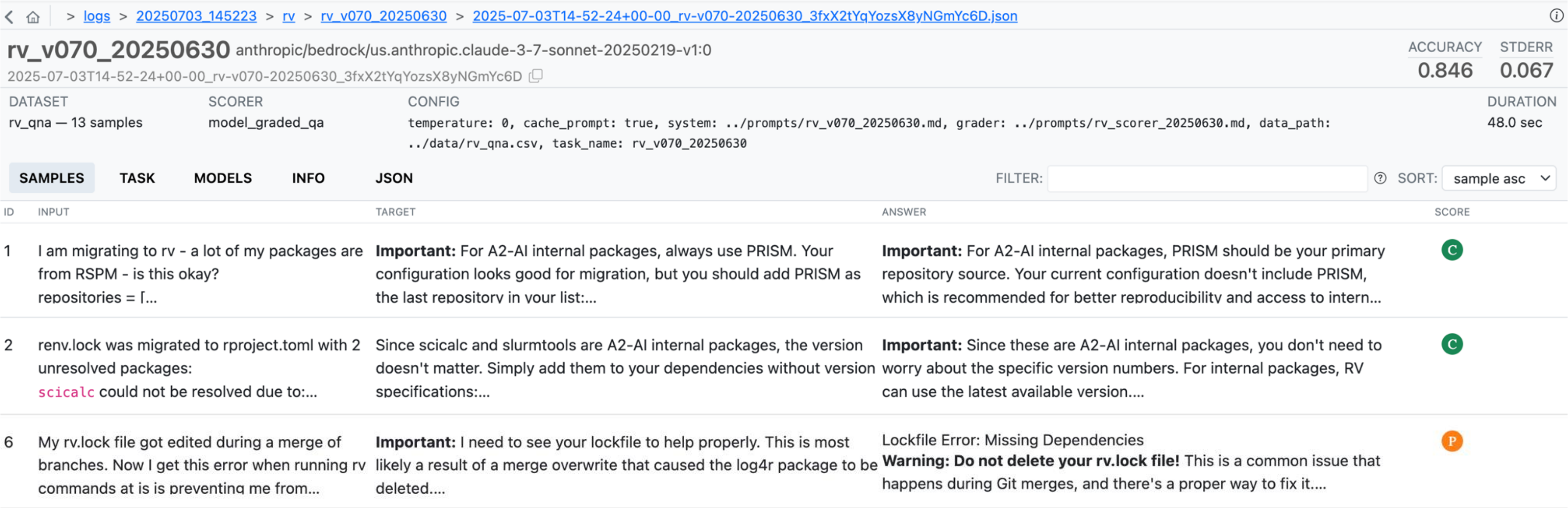


# Evaluation

Does the application ***systematically*** output desired results?

# Evaluation

## Systematically track inputs, outputs, and correct target responses


> logs > 20250703_145223 > rv > rv_v070_20250630 > 2025-07-03T14-52-24+00-00_rv-v070-20250630_3fxX2tYqYozsX8yNGmYc6D.json

**rv_v070_20250630** anthropic/bedrock/us.anthropic.claude-3-7-sonnet-20250219-v1:0

2025-07-03T14-52-24+00-00_rv-v070-20250630_3fxX2tYqYozsX8yNGmYc6D

ACCURACY 0.846

STDERR 0.067

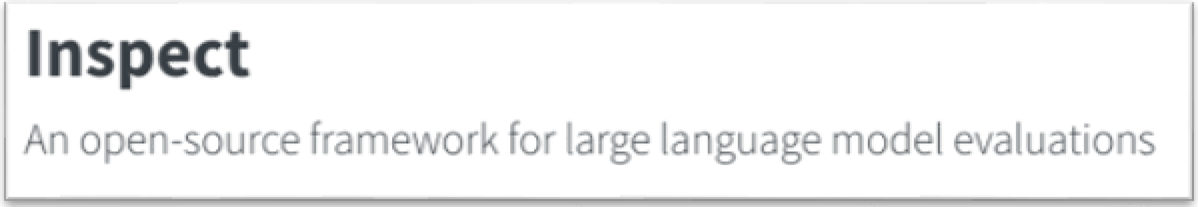
DATASET: rv_qna — 13 samples

SCORER: model_graded_qa

CONFIG: `temperature: 0, cache_prompt: true, system: ../prompts/rv_v070_20250630.md, grader: ../prompts/rv_scorer_20250630.md, data_path: ../data/rv_qna.csv, task_name: rv_v070_20250630`

DURATION: 48.0 sec

| ID | INPUT | TARGET | ANSWER | SCORE |
|---|---|---|---|---|
| 1 | I am migrating to rv - a lot of my packages are from RSPM - is this okay?<br>repositories = [... | **Important:** For A2-AI internal packages, always use PRISM. Your configuration looks good for migration, but you should add PRISM as the last repository in your list:... | **Important:** For A2-AI internal packages, PRISM should be your primary repository source. Your current configuration doesn't include PRISM, which is recommended for better reproducibility and access to intern... | C |
| 2 | renv.lock was migrated to rproject.toml with 2 unresolved packages:<br>`scicalc` could not be resolved due to:... | Since scicalc and slurmtools are A2-AI internal packages, the version doesn't matter. Simply add them to your dependencies without version specifications:... | **Important:** Since these are A2-AI internal packages, you don't need to worry about the specific version numbers. For internal packages, RV can use the latest available version.... | C |
| 6 | My rv.lock file got edited during a merge of branches. Now I get this error when running rv commands at is is preventing me from... | **Important:** I need to see your lockfile to help properly. This is most likely a result of a merge overwrite that caused the log4r package to be deleted.... | Lockfile Error: Missing Dependencies<br>**Warning: Do not delete your rv.lock file!** This is a common issue that happens during Git merges, and there's a proper way to fix it.... | P |



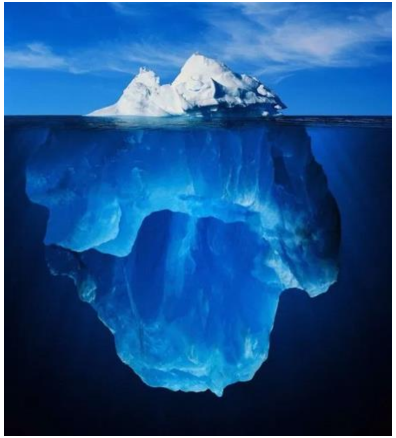
**Inspect**

An open-source framework for large language model evaluations

# Evaluation – option in R

# Takeaways

1. Today’s R GenAI ecosystem is robust enough to do a lot
2. Start building now and refine your requirements as you prototype
3. Bridge strategically if R gaps emerge
   (reticulate, REST APIs, MCP servers)



← model interfaces

← some use cases may require more
(context handling, evaluation…)

http://mailchi.mp/762ea4c14631/are-you-seeing-only-the-tip-of-the-iceberg-move-to-learn-therapy-news?e=d5fdf2d9a8