Package 'rollama'

May 9, 2026

Title Communicate with 'Ollama' to Run Large Language Models Locally

Version 0.3.0

Description Wraps the 'Ollama' <https://ollama.com> API, which can be used to communicate with generative large language models locally.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.3

Imports callr, cli, dplyr, httr2, jsonlite, methods, prettyunits, purrr, rlang, tibble, withr

Suggests base64enc, covr, glue, knitr, rmarkdown, S7, spelling, testthat (>= 3.0.0)

Depends R (>= 4.1.0)

VignetteBuilder knitr

Config/testthat/edition 3

Language en-US

URL https://jbgruber.github.io/rollama/, https://github.com/JBGruber/rollama

BugReports https://github.com/JBGruber/rollama/issues

NeedsCompilation no

Author Johannes B. Gruber [aut, cre] (ORCID: <https://orcid.org/0000-0001-9177-1772>), Maximilian Weber [aut, ctb] (ORCID: <https://orcid.org/0000-0002-1174-449X>)

Maintainer Johannes B. Gruber <JohannesB.Gruber@gmail.com>

Repository CRAN

Date/Publication 2026-03-25 20:30:02 UTC

Contents

`chat_history` *Handle conversations*

Description

Shows and deletes (`new_chat`) the local prompt and response history to start a new conversation.

Usage

```
chat_history()

new_chat()
```

Value

chat_history: tibble with chat history

new_chat: Does not return a value

`check_model_installed` *Check if one or several models are installed on the server*

Description

Check if one or several models are installed on the server

Usage

```
check_model_installed(
  model,
  check_only = FALSE,
  auto_pull = FALSE,
  server = NULL
)
```

Arguments

model	names of one or several models as character vector.
check_only	only return TRUE/FALSE and don't download models.
auto_pull	if FALSE, the default, asks before downloading models.
server	URL to one or several Ollama servers (not the API). Defaults to "http://localhost:11434".

Value

invisible TRUE/FALSE

---

create_model *Create a model from a Modelfile*

Description

Create a model from a Modelfile

Usage

```
create_model(
  model,
  from = NULL,
  template = NULL,
  license = NULL,
  system = NULL,
  parameters = NULL,
  messages = NULL,
  quantize = NULL,
  stream = TRUE,
  ...,
  server = NULL,
  verbose = getOption("rollama_verbose", default = interactive())
)
```

Arguments

`model`	name of the model to create
`from`	existing model to create from
`template`	prompt template to use for the model
`license`	license string or list of licenses for the model
`system`	system prompt to embed in the model
`parameters`	key-value parameters for the model
`messages`	message history to use for the model (array of ChatMessage objects)
`quantize`	quantization level to apply (e.g. `"q4_K_M"`, `"q8_0"`)
`stream`	stream status updates (default: `TRUE`)
`...`	additional arguments (currently unused)
`server`	URL to one or several Ollama servers (not the API). Defaults to "http://localhost:11434".
`verbose`	Whether to print status messages to the Console. Either `TRUE`/`FALSE` or see httr2::progress_bars. The default is to have status messages in interactive sessions. Can be changed with `options(rollama_verbose = FALSE)`.

Details

Custom models are the way to save your system message and model parameters in a dedicated shareable way. If you use `show_model()`, you can look at the configuration of a model in the column modelfile. To get more information and a list of valid parameters, check out `https://docs.ollama.com/modelfile`. Most options are also available through the `query` and `chat` functions, yet are not persistent over sessions.

Value

(invisible) a tibble with information about the created model

Examples

```
create_model("mario", from = "llama3.1", system = "You are mario from Super Mario Bros.")
```

`create_schema` *Create a structured output schema*

Description

Build a JSON Schema for structured output. Pass named type objects as arguments, or supply raw JSON via `.schema`. These functions specify a JSON schema that LLMs can be told to use in their outputs. This is particularly effective for structured data extraction. Their names are based on the JSON schema, which is what the APIs expect behind the scenes. See the structured outputs article for a tutorial.

- `type_boolean()`, `type_integer()`, `type_number()`, and `type_string()` each represent scalars. These are equivalent to length-1 logical, integer, double, and character vectors (respectively).
- `type_enum()` is equivalent to a length-1 factor; it is a string that can only take the specified values.
- `type_array()` is equivalent to a vector in R. You can use it to represent an atomic vector: e.g. `type_array(type_boolean())` is equivalent to a logical vector and `type_array(type_string())` is equivalent to a character vector). You can also use it to represent a list of more complicated types where every element is the same type (R has no base equivalent to this), e.g. `type_array(type_array(type_string()))` represents a list of character vectors.
- `type_object()` is equivalent to a named list in R, but where every element must have the specified type. For example, `type_object(a = type_string(), b = type_array(type_integer()))` is equivalent to a list with an element called `a` that is a string and an element called `b` that is an integer vector.
- `type_ignore()` is used in tool calling to indicate that an argument should not be provided by the LLM. This is useful when the R function has a default value for the argument and you don't want the LLM to supply it.
- `type_from_schema()` allows you to specify the full schema that you want to get back from the LLM as a JSON schema. This is useful if you have a pre-defined schema that you want to use directly without manually creating the type using the `type_*()` functions. You can point to a file with the `path` argument or provide a JSON string with `text`. The schema must be a valid JSON schema object.

Usage

```
create_schema(
  ...,
  .description = NULL,
  .additional_properties = FALSE,
  .schema = NULL
)

type_string(description = NULL, required = TRUE)

type_boolean(description = NULL, required = TRUE)

type_integer(description = NULL, required = TRUE)

type_number(description = NULL, required = TRUE)

type_enum(values, description = NULL, required = TRUE)

type_array(items, description = NULL, required = TRUE)

type_object(
  .description = NULL,
  ...,
```

```
  .required = TRUE,
  .additional_properties = FALSE
)
```

Arguments

`...`	Named rollama type objects representing top-level properties.
`.description, description`	Optional description for the schema.
`.additional_properties`	Whether to allow additional properties.
`.schema`	A raw JSON string or file path to use as the schema directly.
`values`	A character vector of allowed values.
`items`	A rollama type object describing the array items.
`.required, required`	Whether this field is required in the parent object.

See Also

`query()` and `chat()` where the schema is passed via the `format` argument. Structured outputs article for a tutorial.

---

embed_text *Generate Embeddings*

Description

Generate Embeddings

Usage

```
embed_text(
  text,
  model = NULL,
  server = NULL,
  truncate = NULL,
  dimensions = NULL,
  keep_alive = NULL,
  model_params = NULL,
  verbose = getOption("rollama_verbose", default = interactive())
)
```

Arguments

text	text vector to generate embeddings for.
model	which model to use. See https://ollama.com/library for options. Default is "llama3.1". Set option(rollama_model = "modelname") to change default for the current session. See pull_model for more details.
server	URL to one or several Ollama servers (not the API). Defaults to "http://localhost:11434".
truncate	whether to truncate the input to fit within the model's context length (TRUE/FALSE).
dimensions	the desired number of dimensions in the embedding output. Only available for models that support it.
keep_alive	controls how long the model is kept in memory after the request. Accepts a duration string such as "5m" or "1h", 0 to unload immediately, or -1 to keep the model loaded indefinitely.
model_params	a named list of additional model parameters listed in the documentation for the Modelfile.
verbose	Whether to print status messages to the Console (TRUE/FALSE). The default is to have status messages in interactive sessions. Can be changed with options(rollama_verbose = FALSE).

Value

a tibble with embeddings.

Examples

```
## Not run:
embed_text(c(
  "Here is an article about llamas...",
  "R is a language and environment for statistical computing and graphics."))

## End(Not run)
```

---

list_models *List models that are available locally.*

Description

List models that are available locally.

Usage

```
list_models(server = NULL)
```

Arguments

server	URL to one or several Ollama servers (not the API). Defaults to "http://localhost:11434".

Value

a tibble of installed models

`list_running_models` *List running models*

Description

List running models

Usage

```
list_running_models(server = NULL)
```

Arguments

`server`	URL to one or several Ollama servers (not the API). Defaults to "http://localhost:11434".

Value

a tibble of running models

`make_query` *Generate and format queries for a language model*

Description

`make_query` generates structured input for a language model, including system prompt, user messages, and optional examples (assistant answers).

Usage

```
make_query(
  text,
  prompt,
  template = "{prefix}{text}\n{prompt}\n{suffix}",
  system = NULL,
  prefix = NULL,
  suffix = NULL,
  examples = NULL
)
```

Arguments

text	A character vector of texts to be annotated.
prompt	A string defining the main task or question to be passed to the language model.
template	A string template for formatting user queries, containing placeholders like {text}, {prefix}, and {suffix}.
system	An optional string to specify a system prompt.
prefix	A prefix string to prepend to each user query.
suffix	A suffix string to append to each user query.
examples	A tibble with columns text and answer, representing example user messages and corresponding assistant responses.

Details

The function supports the inclusion of examples, which are dynamically added to the structured input. Each example follows the same format as the primary user query.

Value

A list of tibbles, one for each input text, containing structured rows for system messages, user messages, and assistant responses.

Examples

```
template <- "{prefix}{text}\n\n{prompt}{suffix}"
examples <- tibble::tribble(
  ~text, ~answer,
  "This movie was amazing, with great acting and story.", "positive",
  "The film was okay, but not particularly memorable.", "neutral",
  "I found this movie boring and poorly made.", "negative"
)
queries <- make_query(
  text = c("A stunning visual spectacle.", "Predictable but well-acted."),
  prompt = "Classify sentiment as positive, neutral, or negative.",
  template = template,
  system = "Provide a sentiment classification.",
  prefix = "Review: ",
  suffix = " Please classify.",
  examples = examples
)
print(queries)
if (ping_ollama()) { # only run this example when Ollama is running
  query(queries, stream = TRUE, output = "text")
}
```

`ping_ollama` *Ping server to see if Ollama is reachable*

Description

Ping server to see if Ollama is reachable

Usage

```
ping_ollama(server = NULL, silent = FALSE, version = FALSE)
```

Arguments

`server`	URL to one or several Ollama servers (not the API). Defaults to "http://localhost:11434".
`silent`	suppress warnings and status (only return `TRUE`/`FALSE`).
`version`	return version instead of `TRUE`.

Value

TRUE if server is running

`pull_model` *Pull, push, show and delete models*

Description

Pull, push, show and delete models

Usage

```
pull_model(
  model = NULL,
  server = NULL,
  insecure = FALSE,
  background = FALSE,
  verbose = getOption("rollama_verbose", default = interactive())
)

push_model(
  model,
  server = NULL,
  insecure = FALSE,
  verbose = getOption("rollama_verbose", default = interactive())
)
```

```
show_model(model = NULL, detailed = FALSE, server = NULL)

delete_model(model, server = NULL)

copy_model(model, destination = paste0(model, "-copy"), server = NULL)
```

Arguments

model	name of the model(s). Defaults to "llama3.1" when NULL (except in delete_model).
server	URL to one or several Ollama servers (not the API). Defaults to "http://localhost:11434".
insecure	allow insecure connections to the library. Only use this if you are pulling from your own library during development.
background	download model(s) in background without blocking the session.
verbose	Whether to print status messages to the Console. Either TRUE/FALSE or see httr2::progress_bars. The default is to have status messages in interactive sessions. Can be changed with options(rollama_verbose = FALSE).
detailed	when TRUE, the column model_info will contain much more detailed information about the model.
destination	name of the copied model.

Details

- pull_model(): downloads a model from the Ollama registry or Hugging Face
- push_model(): uploads a locally created model to the Ollama registry (ollama.com) so others can pull it. The model name must include your namespace (i.e. "your_username/model_name"). Before pushing you need to add your public key (~/.ollama/id_ed25519.pub) to your ollama.com account settings. This is mainly useful after create_model() — you build a custom model locally (e.g. with a system prompt, quantisation, or fine-tuned weights) and then share it with collaborators or the public. You can also push to a private/self-hosted registry by using a model name that starts with the registry host (e.g. "registry.example.com/mymodel"); set insecure = TRUE if that registry does not use HTTPS.
- show_model(): displays information about a local model
- copy_model(): creates a model with another name from an existing model
- delete_model(): deletes local model

Model names: Model names follow a model:tag format, where model can have an optional namespace such as example/model. Some examples are orca-mini:3b-q4_1 and llama3.1:70b. The tag is optional and, if not provided, will default to latest. The tag is used to identify a specific version.

Value

(invisible) a tibble with information about the model (except in delete_model and push_model)

Note

`push_model()` is intended for advanced users. It requires setup steps that must be completed outside of R: you need an account on https://ollama.com, and your Ollama public key (`~/.ollama/id_ed25519.pub` on Linux/macOS) must be registered in your account settings. The model name must be prefixed with your ollama.com username (e.g. `"your_username/model_name"`); pushing without a namespace will fail with a permission error. Unfortunately, more user-friendly guidance cannot be provided here as the setup process is managed entirely by Ollama outside of R.

Examples

```
## Not run:
# download a model and save information in an object
model_info <- pull_model("mixtral")
# after you pull, you can get the same information with:
model_info <- show_model("mixtral")
# pulling models from Hugging Face Hub is also possible
pull_model("https://huggingface.co/oxyapi/oxy-1-small-GGUF:Q2_K")
# create a custom model and share it on ollama.com
create_model("your_username/mario", from = "llama3.1",
             system = "You are Mario from Super Mario Bros.")
push_model("your_username/mario")

## End(Not run)
```

---

query *Chat with a LLM through Ollama*

Description

Chat with a LLM through Ollama

Usage

```
query(
  q,
  model = NULL,
  stream = TRUE,
  server = NULL,
  images = NULL,
  model_params = NULL,
 output = c("response", "text", "list", "data.frame", "httr2_response", "httr2_request"),
  format = NULL,
  tools = NULL,
  think = NULL,
  keep_alive = NULL,
  logprobs = FALSE,
  top_logprobs = NULL,
  cache = NULL,
```

```
  ...,
  verbose = getOption("rollama_verbose", default = interactive())
)

chat(
  q,
  model = NULL,
  stream = TRUE,
  server = NULL,
  images = NULL,
  model_params = NULL,
  tools = NULL,
  think = NULL,
  keep_alive = NULL,
  logprobs = NULL,
  top_logprobs = NULL,
  ...,
  verbose = getOption("rollama_verbose", default = interactive())
)
```

Arguments

q	the question as a character string or a conversation object.
model	which model(s) to use. See https://ollama.com/library for options. Default is "llama3.1". Set option(rollama_model = "modelname") to change default for the current session. See pull_model for more details.
stream	Logical. Should the answer be printed to the screen.
server	URL to one or several Ollama servers (not the API). Defaults to "http://localhost:11434".
images	path(s) to images (for multimodal models such as llava).
model_params	a named list of additional model parameters listed in the documentation for the Modelfile such as temperature. Use a seed and set the temperature to zero to get reproducible results (see examples).
output	what the function should return. Possible values are "response", "text", "list", "data.frame", "httr2_response" or "httr2_request" or a function see details.
format	the format to return a response in. Use "json" to request arbitrary JSON output or use create_schema() to request a specific structured output. See the structured outputs article for details.
tools	a list of tools (functions) the model may call. Each tool should follow the Ollama tool schema with fields type, function (containing name, description, and parameters).
think	logical. If TRUE, enables extended thinking / reasoning mode (supported by compatible models such as DeepSeek-R1).
keep_alive	controls how long the model is kept in memory after the request. Accepts a duration string such as "5m" or "1h", 0 to unload immediately, or -1 to keep the model loaded indefinitely.
logprobs	logical. If TRUE, the response includes log probabilities of the output tokens.

`top_logprobs` integer (0–20). Number of most-likely tokens to return log probabilities for at each output position. Requires `logprobs = TRUE`.

`cache` where to cache responses on disk so that long annotation pipelines can be resumed after an interruption. Two forms are accepted:

- A **single directory path** (e.g. `"my_cache"`). Each response is stored as `{directory}/{md5_hash}.json`, where the hash is derived from the request content (model, messages, options). Re-running the same request always hits the same file, even across sessions.
- A **character vector** with one explicit file path per request. Use this when you need to control file names yourself.

Existing, valid cache files are loaded instead of re-querying Ollama. Corrupted or missing files are re-requested and then saved. Caching requires `stream = FALSE` (a warning is emitted and streaming is disabled automatically when `cache` is set). The `"httr2_response"` output type and custom output functions are not compatible with caching.

`...` not used.

`verbose` Whether to print status messages to the Console. Either `TRUE`/`FALSE` or see httr2::progress_bars. The default is to have status messages in interactive sessions. Can be changed with `options(rollama_verbose = FALSE)`.

Details

`query` sends a single question to the API, without knowledge about previous questions (only the config message is relevant). `chat` treats new messages as part of the same conversation until new_chat is called.

To make the output reproducible, you can set a seed with `options(rollama_seed = 42)`. As long as the seed stays the same, the models will give the same answer, changing the seed leads to a different answer.

For the output of `query`, there are a couple of options:

- `response`: the response of the Ollama server
- `text`: only the answer as a character vector
- `data.frame`: a data.frame containing model and response
- `list`: a list containing the prompt to Ollama and the response
- `httr2_response`: the response of the Ollama server including HTML headers in the `httr2` response format
- `httr2_request`: httr2_request objects in a list, in case you want to run them with `httr2::req_perform()`, `httr2::req_perform_sequential()`, or `httr2::req_perform_parallel()` yourself.
- a custom function that takes the `httr2_response`(s) from the Ollama server as an input.

Value

list of objects set in output parameter.

Examples

```
# ask a single question
query("why is the sky blue?")

# hold a conversation
chat("why is the sky blue?")
chat("and how do you know that?")

# save the response to an object and extract the answer
resp <- query(q = "why is the sky blue?")
answer <- resp[[1]]$message$content

# or just get the answer directly
answer <- query(q = "why is the sky blue?", output = "text")

# besides the other output options, you can also supply a custom function
query_duration <- function(resp) {
nanosec <- purrr::map(resp, httr2::resp_body_json) |>
  purrr::map_dbl("total_duration")
  round(nanosec * 1e-9, digits = 2)
}
# this function only returns the number of seconds a request took
res <- query("why is the sky blue?", output = query_duration)
res

# ask question about images (to a multimodal model)
images <- c("https://avatars.githubusercontent.com/u/23524101?v=4", # remote
            "/path/to/your/image.jpg") # or local images supported
query(q = "describe these images",
      model = "llava",
      images = images[1]) # just using the first path as the second is not real

# set custom options for the model at runtime (rather than in create_model())
query("why is the sky blue?",
      model_params = list(
        num_keep = 5,
        seed = 42,
        num_predict = 100,
        top_k = 20,
        top_p = 0.9,
        min_p = 0.0,
        tfs_z = 0.5,
        typical_p = 0.7,
        repeat_last_n = 33,
        temperature = 0.8,
        repeat_penalty = 1.2,
        presence_penalty = 1.5,
        frequency_penalty = 1.0,
        mirostat = 1,
        mirostat_tau = 0.8,
        mirostat_eta = 0.6,
        penalize_newline = TRUE,
```

```
        numa = FALSE,
        num_ctx = 1024,
        num_batch = 2,
        num_gpu = 0,
        main_gpu = 0,
        low_vram = FALSE,
        vocab_only = FALSE,
        use_mmap = TRUE,
        use_mlock = FALSE,
        num_thread = 8
      ))

# use a seed to get reproducible results
query("why is the sky blue?", model_params = list(seed = 42))

# to set a seed for the whole session you can use
options(rollama_seed = 42)

# this might be interesting if you want to turn off the GPU and load the
# model into the system memory (slower, but most people have more RAM than
# VRAM, which might be interesting for larger models)
query("why is the sky blue?",
       model_params = list(num_gpu = 0))

# enable extended thinking / reasoning mode (supported models e.g. DeepSeek-R1)
query("what is 3 * 12?", model = "deepseek-r1", think = TRUE)

# use tools (function calling) — tool calling is a two-step process:
# 1. The model returns a tool_call (empty content) instead of a text answer.
# 2. You execute the function and send the result back so the model can
#    formulate a final answer.

# define the actual R function
add_numbers <- function(a, b) as.numeric(a) + as.numeric(b)

# describe it to the model
tools <- list(list(
  type = "function",
  `function` = list(
    name = "add_numbers",
    description = "Add two numbers together",
    parameters = list(
      type = "object",
      properties = list(
        a = list(type = "number", description = "First number"),
        b = list(type = "number", description = "Second number")
      ),
      required = list("a", "b")
    )
  )
))

# Step 1: model decides which tool to call and with which arguments
```

```
question <- "What is 4 + 7?"
resp <- query(question, model = "llama3.1", tools = tools, stream = FALSE)
tool_call <- resp[[1]]$message$tool_calls[[1]]

# Step 2: call the real function with the model-supplied arguments
result <- do.call(add_numbers, tool_call$`function`$arguments)

# Step 3: send the result back so the model can give a final answer
conversation <- data.frame(
  role    = c("user", "assistant", "tool"),
  content = c(question, "", as.character(result))
)
query(conversation, model = "llama3.1")

# Asking the same question to multiple models is also supported
query("why is the sky blue?", model = c("llama3.1", "orca-mini"))

# And if you have multiple Ollama servers in your network, you can send
# requests to them in parallel
if (ping_ollama(c("http://localhost:11434/",
                  "http://192.168.2.45:11434/"))) { # check if servers are running
  query("why is the sky blue?", model = c("llama3.1", "orca-mini"),
        server = c("http://localhost:11434/",
                   "http://192.168.2.45:11434/"))
}
```

`rollama-options` *rollama Options*

Description

The behaviour of `rollama` can be controlled through `options()`. Specifically, the options below can be set.

Details

rollama_server This controls the default server where Ollama is expected to run. It assumes that you are running Ollama locally in a Docker container.

`"http://localhost:11434"`

default rollama_model The default model is llama3.1, which is a good overall option with reasonable performance and size for most tasks. You can change the model in each function call or globally with this option.

`"llama3.1"`

default rollama_verbose Whether the package tells users what is going on, e.g., showing a spinner while the models are thinking or showing the download speed while pulling models. Since this adds some complexity to the code, you might want to disable it when you get errors (it won't fix the error, but you get a better error trace).

`TRUE`

default_rollama_config The default configuration or system message. If NULL, the system message defined in the used model is employed.

None

default_rollama_seed As long as the seed stays the same, the models will give the same answer, changing the seed leads to a different answer. Per default, no seed is set and each call to `query()` or `chat()` will give you a different answer.

None

Examples

default: `options(rollama_config = "You make answers understandable to a 5 year old")`

Index