Package: rlmstudio (via r-universe)

November 20, 2026

Title Access and Control LM Studio

Version 0.4.0

Description Run local Large Language Models (LLMs) over many texts from 'R' without sending data to a third party. A community-maintained wrapper for the 'LM Studio' command line interface and API that provides functions to manage the local daemon and server, download and load models, and score, label, or generate text at scale.

License MIT + file LICENSE

URL https://jmgirard.github.io/rlmstudio/, https://github.com/jmgirard/rlmstudio

BugReports https://github.com/jmgirard/rlmstudio/issues

Depends R (>= 4.1.0)

Imports cli, httr2, jsonlite, processx, utils

Suggests httptest2, httpuv, knitr, mockery, rmarkdown, testthat (>= 3.3.0), withr

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Jeffrey Girard [aut, cph, cre] (ORCID: <https://orcid.org/0000-0002-7359-3746>)

Maintainer Jeffrey Girard <me@jmgirard.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-11-20 21:21:34 UTC

RemoteUrl https://github.com/cran/rlmstudio

RemoteRef HEAD

RemoteSha 766c249ad4d864d9e5bbd01e7893b5768431dc0f

Contents

check_lms_version . . . 2
has_lms . . . 3
install_lmstudio . . . 3
list_instances . . . 4
list_models . . . 9
lms_chat . . . 13
lms_chat_batch . . . 23
lms_chat_native . . . 33
lms_chat_openai . . . 38
lms_chat_openresponses . . . 48
lms_daemon_start . . . 56
lms_daemon_status . . . 57
lms_daemon_stop . . . 57
lms_download . . . 58
lms_download_status . . . 62
lms_embed . . . 66
lms_load . . . 72
lms_path . . . 77
lms_score_expected . . . 78
lms_server_ready . . . 79
lms_server_start . . . 81
lms_server_status . . . 83
lms_server_stop . . . 84
lms_unload . . . 85
lms_unload_all . . . 88
rlmstudio-conditions . . . 91
rlmstudio_token . . . 98
with_lms_daemon . . . 100

Index **101**

`check_lms_version` *Check if the installed LM Studio CLI meets the minimum requirement*

Description

Check if the installed LM Studio CLI meets the minimum requirement

Usage

```
check_lms_version(min_version = "0.4.0")
```

Arguments

`min_version`	Character string of the required version. Default is "0.4.0".

Value

A logical scalar: TRUE if the LM Studio CLI version meets or exceeds the specified min_version, and FALSE otherwise, including when lms_path() does not find the CLI.

Examples

```
## Not run:
check_lms_version("0.4.0")

## End(Not run)
```

---

has_lms *Check if LM Studio CLI is installed*

Description

Uses the same lookup as lms_path(): the RLMSTUDIO_LMS_PATH environment variable, then the system PATH, then common installation directories.

Usage

```
has_lms()
```

Value

A logical scalar: TRUE if lms_path() finds the lms executable, and FALSE if it aborts.

Examples

```
## Not run:
has_lms()

## End(Not run)
```

---

install_lmstudio *Help the user install or update LM Studio*

Description

This function provides two methods for setting up LM Studio on your system. The "browser" method opens the official download page for the LM Studio desktop application (GUI). The "headless" method runs an automated installation script to install the llmster daemon and CLI, which is suitable for servers, containers, or users who prefer a GUI-less environment.

Usage

```
install_lmstudio(method = c("browser", "headless"))
```

Arguments

`method`	Character. Either "browser" (opens the GUI download page) or "headless" (installs the `llmster` daemon via script).

Details

If the headless installer exits with a status other than 0, the function aborts. The message gives the exit code and quotes the installer output, after "The installer said:". A byte that is not valid UTF-8 shows as <xx>, its hex value. ANSI escape codes, such as color codes, cursor codes, and terminal links, are removed, and each run of whitespace becomes one space. A text longer than 1000 characters keeps at most its last 1000 characters, after "...". Any other error of the install step, such as a missing `curl`, aborts with "Headless installation failed." and the error message.

Value

Invisibly returns `TRUE` upon successful completion. This function is primarily utilized for its side effects of opening a web browser or executing system installation commands.

Examples

```
## Not run:
# Open your default web browser to the download page
install_lmstudio(method = "browser")

# Attempt automatic headless installation via the command line
install_lmstudio(method = "headless")

## End(Not run)
```

`list_instances` *List loaded model instances*

Description

Retrieves the loaded model instances on the server via the LM Studio REST API, one row per instance, with the load configuration of each instance in columns. It is the view that `lms ps` prints, read from the same model list that `list_models()` reads, so it honors `host` and `token`.

Usage

```
list_instances(
  type = c("llm", "embedding"),
  quiet = NULL,
  host = "http://localhost:1234",
  token = NULL
)
```

Arguments

type	Character vector. The types of models to include. Defaults to `c("llm", "embedding")`. It must hold one or more elements, and no element can be `NA`, empty, or whitespace only. Each element must be valid in its declared encoding and not marked `"bytes"`. Any other value, `NULL` and a factor included, aborts before the check for a running server. A type that no model has, such as `"vlm"`, matches nothing. A classed filter matches by its value, and a class method such as `as.character()` does not change the match.
quiet	`TRUE`, `FALSE`, or `NULL`, the default. `NULL` follows the `rlmstudio.quiet` option. `TRUE` hides the message printed when no instance is found, and `FALSE` prints it, also when the option is `TRUE`. Any other value, `NA` included, aborts before the check for a running server. Does not suppress the abort raised when the server is not running.
host	Character. The host address of the local server.
token	Character or `NULL`. An API token for a server that requires authentication. `NULL` reads the `rlmstudio.token` option and then the `RLMSTUDIO_API_TOKEN` environment variable. See rlmstudio_token.

Details

The model list does not carry four fields that `lms ps --json` reports: the generation status, the queued requests, the ttl setting, and the last-used time. It does carry `remaining_ttl_seconds` for an instance loaded with a ttl, and `list_instances()` returns it in the `remaining_ttl_seconds` column.

Value

A `data.frame` with one row per loaded instance of a model whose type is in `type`, with the columns that the "Columns" section describes. If there is no such instance, it returns a `data.frame` with zero rows and the five fixed columns, four character columns and the double column `remaining_ttl_seconds`, invisibly, and prints a message that `quiet` controls.

Columns

The first five columns are the fixed columns. The first four are character columns:

- `id`: the id of the instance.
- `key`: the key of its model.
- `type`: the type of its model, such as `"llm"` or `"embedding"`.
- `display_name`: the display name of its model, or `NA` when the model list gives none or `null`. An empty name stays `""`.

The fifth is a double column:

- `remaining_ttl_seconds`: the `remaining_ttl_seconds` that the model list gives for the instance, or `NA` when the field is absent or `null`. In LM Studio 0.4.26+4, the value counted down from the ttl of a load with `lms load --ttl`, and an instance loaded with no ttl had no such field, so it holds `NA`. No other LM Studio version was checked for this field.

Rows follow the order of the models in the model list, then the order of the instances of a model. Two instances with the same id, or two models with the same key, give separate rows.

After the five columns, the frame has one column for each field name in the `config` object of an instance, in order of first appearance. When one `config` holds a name twice, the first value counts. A row whose `config` lacks the field, or has no `config`, holds `NA` there, or `NULL` in a list-column.

A column takes the field name as it is, with no name repair, so a name such as `a-b` needs backticks or `[[`. A field name that is empty, or that equals one of the five fixed column names or the column name of an earlier field, takes the prefix `config.`, again and again while the name still clashes. A field named `id` so gives the column `config.id`, and a field named `remaining_ttl_seconds` gives `config.remaining_ttl_seconds`.

The column type follows the values of the field that are present and not `null`:

- All strings give a character column.
- All numbers give a double column, whole numbers included.
- All booleans give a logical column.
- No such values give a logical column of `NA`.

In these four kinds, a `null` value is `NA`. Any other mix, or any JSON object or array, gives a list-column. It holds each value as `jsonlite::parse_json()` returns it, and `NULL` where the field is absent or `null`.

Server not running

Functions that call the LM Studio REST API open a TCP connection to the hostname and port named in `host` before they send the request. A function that checks its own arguments does that first, so a bad `model`, `job_id`, `input`, `inputs`, `messages`, `schema`, `ttl`, `previous_response_id`, or `batch_size`, a bad `type` of `list_models()` or `list_instances()`, a bad `TRUE` or `FALSE` argument such as `simplify`, `logprobs`, `quiet`, `force`, or `store`, a `store` on the `"openai"` route of `lms_chat()` or `lms_chat_batch()`, a `stream` in the `...` of a chat function, a `logprobs` in the `...` of `lms_chat_batch()` or `lms_chat_native()`, a value in the `...` of `lms_chat_batch()` that `lms_chat()` reads as an argument already given, an `instructions` or `messages` in the `...` of `lms_chat()` or `lms_chat_batch()` on the route where `lms_chat()` sets it, or a name, id, or `type` string that is not valid in its encoding or is marked as bytes, aborts with an argument message and no condition class even when the server is down. A condition of class `rlmstudio_no_server` is raised when that connection cannot be opened. A refused connection raises it. So do an address the package cannot parse and a hostname that does not resolve. An address that neither accepts nor refuses the connection also raises it. That case waits for the operating system to give up, which can take a minute. If the hostname of `host` is `localhost`, in any letter case, or `127.0.0.1`, start the server with `lms_server_start()` and the port of `host`. For the default `host`, that is `lms_server_start(port = 1234)`. With no `port`, LM Studio uses the port that it used last. For any other `host`, start the server on the machine that `host` names, or give `host` the address that your server listens on. A `host` that does not parse gets this second hint too. The condition message gives the same hint for the `host` of the call.

The check reads the port and nothing else. Any process holding that port accepts the connection, so the condition is not raised even though no LM Studio server is there. The call then does not raise `rlmstudio_no_server`, and what it does depends on what answers. On a status-200 body that does not parse as JSON, the chat functions, `lms_embed()`, `list_models()`, `list_instances()`,

lms_load(), lms_download(), lms_download_status(), and lms_unload_all() raise rlmstudio_bad_response. lms_unload() does not read the body, so it can report success. A body that parses as JSON but has another shape can come back unchanged with simplify = FALSE. With simplify = TRUE, the chat functions and lms_embed() raise rlmstudio_bad_response for it. list_models() and list_instances() raise it for a model list with another shape, and so do lms_unload_all() and lms_load() without force = TRUE, which read that list. lms_chat_openai() and lms_chat_openresponses() raise it for such a model list too, with either setting of simplify, through the model lookup that a reply from another model starts. lms_chat() raises it through them. lms_embed() raises it in the same way for the inputs of that request. lms_load(), lms_download(), and lms_download_status() raise it for a reply of their own with another shape, such as {}. A process that does not answer in HTTP gives an httr2_failure error. Use lms_server_ready() for the stronger test: it asks the host for a model list and reports TRUE only for a model list that list_models() can read.

lms_chat_batch() checks the server once before its first input, and lms_chat() checks it again for each input. If that check finds the server gone during the batch, the batch aborts with rlmstudio_no_server, and no request goes out after that. The condition then carries a results field, a list as long as inputs. Its elements before the lost input hold the values that format = "list" returns for those inputs. The element of the lost input and every element after it are NULL. The check before the first input adds no results field. A connection that fails after the check passes, such as a server that stops during a request, raises an httr2_failure error instead. That error aborts the batch and carries no results field.

lms_embed() checks the server before each request, and a request carries at most batch_size inputs. If a check after the first request, or the check of a model-list lookup, finds the server gone, the call aborts with rlmstudio_no_server. Once a request has succeeded, the condition carries a results field. With simplify = TRUE, results is a matrix with NA in each row whose embedding did not arrive. With simplify = FALSE, it is a list with one element per batch, with NULL in the element of the request that ended the call and in every element after it.

API failure

A condition of class rlmstudio_api_error is raised when a REST call returns a response that the wrapper treats as a failure. The condition carries a status field, which holds the HTTP response status as an integer. It also carries a code field. The field holds the string at error.code of the response body, such as "model_not_found". It is NULL when the body does not parse, when error is not a JSON object, or when its code is not one string. A server_message field holds the error text as the server sent it: the error string of the body, or else the string at error.message. It is NULL when the body does not parse or holds neither string.

On LM Studio 0.4.26+4 with no model loaded, /v1/chat/completions, /v1/responses, and /v1/embeddings answered a model name that the server could not find with status 400, no code, and a server_message that starts with "No models loaded".

lms_chat_batch() aborts on it when its status is 401, 403, or 404, when its status is 400 and its code is "model_not_found", and when its status is 400 and its server_message starts with "No models loaded", in that letter case. No request goes out after that input. The condition then carries a results field that follows the rule for a lost server in the "Server not running" section: its elements before the failed input hold the values that format = "list" returns for those inputs, and the element of the failed input and every element after it are NULL. For any other status, the element of the failed input holds the condition, or NA where the result is text, and the batch warns once and goes on. See the details of lms_chat_batch().

lms_embed() follows the same rule for each request, except for the code "model_not_found". A 401, 403, or 404 aborts the call, and so does a 400 whose server_message starts with "No models loaded". Once a request has succeeded, the condition carries a results field, a matrix or a list as the "Server not running" section describes. Any other status, a 400 with the code "model_not_found" included, fails the inputs of that request alone, and the call warns once and goes on. If every request fails, the call aborts with the first condition. See the details of lms_embed().

Malformed response

A condition of class rlmstudio_bad_response is raised when the server answers with a status the wrapper accepts and a body the wrapper cannot read. It is raised where a wrapper checks the body before it reshapes it, rather than indexing straight into whatever arrived. Eleven functions raise it for a body they cannot read: lms_embed(), lms_chat(), lms_chat_native(), lms_chat_openresponses(), lms_chat_openai(), list_models(), list_instances(), lms_load(), lms_download(), lms_download_status(), and lms_unload_all(). lms_chat() raises it through the chat function it calls. lms_unload_all() raises it through list_models(), and so does lms_load() unless force = TRUE. lms_chat_batch() raises it only as rlmstudio_model_mismatch, which the "Reply from another model" section of lms_chat_openai() describes.

All eleven raise it for a status-200 body that does not parse as JSON, such as an HTML page from a proxy, JSON text that stops part way, or an empty body. In the functions that take simplify, the body is parsed before simplify is read, so the condition is raised whatever simplify is. The body is parsed by its content and not by its Content-Type header, so valid JSON under text/plain is read as JSON. The body is read as JSON text and nothing else. A body whose text is a URL or the path of a file does not parse, and the package does not fetch the URL or read the file. The message says that the body did not parse as JSON and that something other than LM Studio may be answering on the host. It does not hold the body text.

The condition carries a status field, which holds the HTTP response status as an integer. Today the status is always 200: each of these functions reads the body only after a 200, and reports every other status as an rlmstudio_api_error instead.

Malformed model list

list_models() also raises it for a status-200 model list with the wrong shape. lms_unload_all() and lms_load() without force = TRUE raise it through list_models(). lms_chat_openai(), lms_chat_openresponses(), and lms_embed() apply these rules to the model list of their model lookup, as the "Reply from another model" section of lms_chat_openai() describes. A model list must follow four rules. Each field is read by its exact name, so a field named keyX does not stand in for key.

1. The body is a JSON object whose models field is an array. The array can be empty.
2. Each entry of models is a JSON object. Its type and key are strings, and its loaded_instances is an array.
3. The size_bytes of an entry is a number, or absent, or null. So is its max_context_length.
4. Each entry of loaded_instances is a JSON object whose id is a string with a character that is not whitespace.

The rules are checked before the type and loaded filters, so an entry that the filters drop can still raise the condition. The message names the field or entry that broke a rule. lms_server_ready() applies the same rules and returns FALSE for a body that breaks one.

`list_instances()` applies the four rules too, and three more. They cover only an entry whose `type` is in its `type` argument and that has at least one loaded instance. The `display_name` of such an entry is a string, or absent, or `null`. The `remaining_ttl_seconds` of each of its instances is a number, or absent, or `null`. The `config` of each of its instances is a JSON object, or absent, or `null`. `list_models()` and `lms_server_ready()` do not apply these three rules.

See Also

LM Studio List Models API, and `list_models()` for one row per model.

Examples

```
## Not run:
lms_server_start(port = 1234)
lms_load("google/gemma-3-1b")

# One row per loaded instance, with its load configuration
list_instances()

# Only the loaded embedding models
list_instances(type = "embedding")

## End(Not run)
```

---

list_models *List available models*

Description

Retrieves a list of models available on your system via the LM Studio REST API.

Usage

```
list_models(
  loaded = FALSE,
  type = c("llm", "embedding"),
  detailed = FALSE,
  quiet = NULL,
  host = "http://localhost:1234",
  token = NULL
)
```

Arguments

`loaded`	`TRUE` or `FALSE`. If `TRUE`, returns only currently loaded models. Defaults to `FALSE`. Any other value, `NULL` and `NA` included, aborts before the check for a running server.

type	Character vector. The types of models to include. Defaults to `c("llm", "embedding")`. It must hold one or more elements, and no element can be `NA`, empty, or whitespace only. Each element must be valid in its declared encoding and not marked `"bytes"`. Any other value, `NULL` and a factor included, aborts before the check for a running server. A type that no model has, such as `"vlm"`, matches nothing. A classed filter matches by its value, and a class method such as `as.character()` does not change the match.
detailed	`TRUE` or `FALSE`. Show all information about each model. Defaults to `FALSE`. Any other value, `NULL` and `NA` included, aborts before the check for a running server.
quiet	`TRUE`, `FALSE`, or `NULL`, the default. `NULL` follows the `rlmstudio.quiet` option. `TRUE` hides the message printed when no model is found, and `FALSE` prints it, also when the option is `TRUE`. Any other value, `NA` included, aborts before the check for a running server. Does not suppress the abort raised when the server is not running.
host	Character. The host address of the local server.
token	Character or `NULL`. An API token for a server that requires authentication. `NULL` reads the `rlmstudio.token` option and then the `RLMSTUDIO_API_TOKEN` environment variable. See rlmstudio_token.

Value

A `data.frame` containing information about the available models. By default, it includes columns for `state`, `type`, `display_name`, `key`, `architecture`, and `size_gb`. If `detailed = TRUE`, it returns a comprehensive `data.frame` including all raw metadata columns provided by the API. Returns an empty `data.frame` if no models match the criteria.

Server not running

Functions that call the LM Studio REST API open a TCP connection to the hostname and port named in `host` before they send the request. A function that checks its own arguments does that first, so a bad `model`, `job_id`, `input`, `inputs`, `messages`, `schema`, `ttl`, `previous_response_id`, or `batch_size`, a bad `type` of `list_models()` or `list_instances()`, a bad `TRUE` or `FALSE` argument such as `simplify`, `logprobs`, `quiet`, `force`, or `store`, a `store` on the `"openai"` route of `lms_chat()` or `lms_chat_batch()`, a `stream` in the `...` of a chat function, a `logprobs` in the `...` of `lms_chat_batch()` or `lms_chat_native()`, a value in the `...` of `lms_chat_batch()` that `lms_chat()` reads as an argument already given, an `instructions` or `messages` in the `...` of `lms_chat()` or `lms_chat_batch()` on the route where `lms_chat()` sets it, or a name, id, or `type` string that is not valid in its encoding or is marked as bytes, aborts with an argument message and no condition class even when the server is down. A condition of class `rlmstudio_no_server` is raised when that connection cannot be opened. A refused connection raises it. So do an address the package cannot parse and a hostname that does not resolve. An address that neither accepts nor refuses the connection also raises it. That case waits for the operating system to give up, which can take a minute. If the hostname of `host` is `localhost`, in any letter case, or `127.0.0.1`, start the server with `lms_server_start()` and the port of `host`. For the default `host`, that is `lms_server_start(port = 1234)`. With no `port`, LM Studio uses the port that it used last. For any other `host`, start the server on the machine that `host` names, or give `host` the address that your server listens on. A `host` that does not parse gets this second hint too. The condition message gives the same hint for the `host` of the call.

The check reads the port and nothing else. Any process holding that port accepts the connection, so the condition is not raised even though no LM Studio server is there. The call then does not raise `rlmstudio_no_server`, and what it does depends on what answers. On a status-200 body that does not parse as JSON, the chat functions, `lms_embed()`, `list_models()`, `list_instances()`, `lms_load()`, `lms_download()`, `lms_download_status()`, and `lms_unload_all()` raise `rlmstudio_bad_response`. `lms_unload()` does not read the body, so it can report success. A body that parses as JSON but has another shape can come back unchanged with `simplify = FALSE`. With `simplify = TRUE`, the chat functions and `lms_embed()` raise `rlmstudio_bad_response` for it. `list_models()` and `list_instances()` raise it for a model list with another shape, and so do `lms_unload_all()` and `lms_load()` without `force = TRUE`, which read that list. `lms_chat_openai()` and `lms_chat_openresponses()` raise it for such a model list too, with either setting of `simplify`, through the model lookup that a reply from another model starts. `lms_chat()` raises it through them. `lms_embed()` raises it in the same way for the inputs of that request. `lms_load()`, `lms_download()`, and `lms_download_status()` raise it for a reply of their own with another shape, such as `{}`. A process that does not answer in HTTP gives an `httr2_failure` error. Use `lms_server_ready()` for the stronger test: it asks the host for a model list and reports `TRUE` only for a model list that `list_models()` can read.

`lms_chat_batch()` checks the server once before its first input, and `lms_chat()` checks it again for each input. If that check finds the server gone during the batch, the batch aborts with `rlmstudio_no_server`, and no request goes out after that. The condition then carries a `results` field, a list as long as `inputs`. Its elements before the lost input hold the values that `format = "list"` returns for those inputs. The element of the lost input and every element after it are `NULL`. The check before the first input adds no `results` field. A connection that fails after the check passes, such as a server that stops during a request, raises an `httr2_failure` error instead. That error aborts the batch and carries no `results` field.

`lms_embed()` checks the server before each request, and a request carries at most `batch_size` inputs. If a check after the first request, or the check of a model-list lookup, finds the server gone, the call aborts with `rlmstudio_no_server`. Once a request has succeeded, the condition carries a `results` field. With `simplify = TRUE`, `results` is a matrix with `NA` in each row whose embedding did not arrive. With `simplify = FALSE`, it is a list with one element per batch, with `NULL` in the element of the request that ended the call and in every element after it.

API failure

A condition of class `rlmstudio_api_error` is raised when a REST call returns a response that the wrapper treats as a failure. The condition carries a `status` field, which holds the HTTP response status as an integer. It also carries a `code` field. The field holds the string at `error.code` of the response body, such as `"model_not_found"`. It is `NULL` when the body does not parse, when `error` is not a JSON object, or when its `code` is not one string. A `server_message` field holds the error text as the server sent it: the `error` string of the body, or else the string at `error.message`. It is `NULL` when the body does not parse or holds neither string.

On LM Studio 0.4.26+4 with no model loaded, `/v1/chat/completions`, `/v1/responses`, and `/v1/embeddings` answered a model name that the server could not find with status 400, no `code`, and a `server_message` that starts with "No models loaded".

`lms_chat_batch()` aborts on it when its `status` is 401, 403, or 404, when its `status` is 400 and its `code` is `"model_not_found"`, and when its `status` is 400 and its `server_message` starts with "No models loaded", in that letter case. No request goes out after that input. The condition then carries a `results` field that follows the rule for a lost server in the "Server not running" section: its

elements before the failed input hold the values that `format = "list"` returns for those inputs, and the element of the failed input and every element after it are `NULL`. For any other status, the element of the failed input holds the condition, or `NA` where the result is text, and the batch warns once and goes on. See the details of `lms_chat_batch()`.

`lms_embed()` follows the same rule for each request, except for the code `"model_not_found"`. A 401, 403, or 404 aborts the call, and so does a 400 whose `server_message` starts with "No models loaded". Once a request has succeeded, the condition carries a `results` field, a matrix or a list as the "Server not running" section describes. Any other status, a 400 with the code `"model_not_found"` included, fails the inputs of that request alone, and the call warns once and goes on. If every request fails, the call aborts with the first condition. See the details of `lms_embed()`.

Malformed response

A condition of class `rlmstudio_bad_response` is raised when the server answers with a status the wrapper accepts and a body the wrapper cannot read. It is raised where a wrapper checks the body before it reshapes it, rather than indexing straight into whatever arrived. Eleven functions raise it for a body they cannot read: `lms_embed()`, `lms_chat()`, `lms_chat_native()`, `lms_chat_openresponses()`, `lms_chat_openai()`, `list_models()`, `list_instances()`, `lms_load()`, `lms_download()`, `lms_download_status()`, and `lms_unload_all()`. `lms_chat()` raises it through the chat function it calls. `lms_unload_all()` raises it through `list_models()`, and so does `lms_load()` unless `force = TRUE`. `lms_chat_batch()` raises it only as `rlmstudio_model_mismatch`, which the "Reply from another model" section of `lms_chat_openai()` describes.

All eleven raise it for a status-200 body that does not parse as JSON, such as an HTML page from a proxy, JSON text that stops part way, or an empty body. In the functions that take `simplify`, the body is parsed before `simplify` is read, so the condition is raised whatever `simplify` is. The body is parsed by its content and not by its `Content-Type` header, so valid JSON under `text/plain` is read as JSON. The body is read as JSON text and nothing else. A body whose text is a URL or the path of a file does not parse, and the package does not fetch the URL or read the file. The message says that the body did not parse as JSON and that something other than LM Studio may be answering on the host. It does not hold the body text.

The condition carries a `status` field, which holds the HTTP response status as an integer. Today the status is always 200: each of these functions reads the body only after a 200, and reports every other status as an `rlmstudio_api_error` instead.

Malformed model list

`list_models()` also raises it for a status-200 model list with the wrong shape. `lms_unload_all()` and `lms_load()` without `force = TRUE` raise it through `list_models()`. `lms_chat_openai()`, `lms_chat_openresponses()`, and `lms_embed()` apply these rules to the model list of their model lookup, as the "Reply from another model" section of `lms_chat_openai()` describes. A model list must follow four rules. Each field is read by its exact name, so a field named `keyX` does not stand in for `key`.

1. The body is a JSON object whose `models` field is an array. The array can be empty.
2. Each entry of `models` is a JSON object. Its `type` and `key` are strings, and its `loaded_instances` is an array.
3. The `size_bytes` of an entry is a number, or absent, or `null`. So is its `max_context_length`.

4. Each entry of `loaded_instances` is a JSON object whose `id` is a string with a character that is not whitespace.

The rules are checked before the `type` and `loaded` filters, so an entry that the filters drop can still raise the condition. The message names the field or entry that broke a rule. `lms_server_ready()` applies the same rules and returns `FALSE` for a body that breaks one.

`list_instances()` applies the four rules too, and three more. They cover only an entry whose `type` is in its `type` argument and that has at least one loaded instance. The `display_name` of such an entry is a string, or absent, or `null`. The `remaining_ttl_seconds` of each of its instances is a number, or absent, or `null`. The `config` of each of its instances is a JSON object, or absent, or `null`. `list_models()` and `lms_server_ready()` do not apply these three rules.

See Also

LM Studio List Models API, and `list_instances()` for one row per loaded instance.

Examples

```
## Not run:
lms_server_start(port = 1234)
lms_download("google/gemma-3-1b")
lms_load("google/gemma-3-1b")

# List all downloaded models
list_models()

# List only currently loaded models
list_models(loaded = TRUE)

# Get detailed information about loaded text models
list_models(loaded = TRUE, type = "llm", detailed = TRUE)

## End(Not run)
```

`lms_chat` *Chat Completion with LM Studio*

Description

Send a prompt to a locally running LM Studio model. This wrapper automatically routes your request to the appropriate subfunction based on the selected API type.

Usage

```
lms_chat(
  model,
  input,
  system_prompt = NULL,
  host = "http://localhost:1234",
```

```
  api_type = c("openresponses", "openai", "native"),
  logprobs = FALSE,
  simplify = TRUE,
  ...,
  schema = NULL,
  ttl = NULL,
  previous_response_id = NULL,
  store = NULL,
  token = NULL
)
```

Arguments

model	Character. The name of the loaded model. Must be one name, given as a single string. The string must be valid in its declared encoding and not marked "bytes". A class, names, and the S4 bit are removed before the name is sent.
input	Character. The user prompt to send to the model, as one string. A character vector must hold no missing values, and its length must be one. Any other character value aborts before the check for a running server. To send several prompts, one request each, use lms_chat_batch(). A list is sent as given, as the input field or, on the "openai" route, as the content of the user message.
system_prompt	Character. An optional system prompt to guide model behavior.
host	Character. The base URL of the LM Studio server. Default is "http://localhost:1234".
api_type	Character. The LM Studio API endpoint to use. Options are "openresponses" (default), "openai", or "native".
logprobs	TRUE or FALSE. Whether to return the log probabilities of the generated tokens. Default is FALSE. Any other value, NULL and NA included, aborts before the check for a running server.
simplify	TRUE or FALSE. If TRUE, extracts the core text response. Default is TRUE. Any other value, NULL and NA included, aborts before the check for a running server.
...	Additional arguments passed to the selected API body. The package checks a stream here. A stream other than FALSE or NULL aborts before the call checks for a running server, because the package reads a whole reply and not a streamed one. An instructions here with api_type = "openresponses", the default, aborts before the request, because lms_chat() sets instructions from system_prompt on that route. A messages here with api_type = "openai" also aborts, because lms_chat() builds messages from system_prompt and input on that route. The message names the argument, and the abort has no condition class. Only these exact names abort. On the other routes, each goes into the request body.
schema	A JSON Schema that the reply must match, or NULL. It needs api_type = "openai", and any other api_type aborts before the request. See lms_chat_openai() for its form and for what is returned.
ttl	A whole number of seconds from 1 to .Machine$integer.max, or NULL. It is how long the model stays loaded with no request. It needs api_type =

"openai", and any other api_type, the default included, aborts before the request. It has an effect only on a model that this request loads. The server loads a model that is not loaded yet when its just-in-time loading setting is on. A model that is already loaded keeps its idle time.

previous_response_id

One string, a value that carries a response_id attribute, or NULL. It names the stored reply that this chat continues. Pass the earlier reply itself, such as first after first <- lms_chat(...), or its id, attr(first, "response_id"). A value that carries a response_id attribute sends that attribute in place of the value, also when the value is a list or is itself an id. Only that exact attribute name is read. A value with no such attribute is sent as the id itself. So a reply that came back with no id, such as a native reply sent with store = FALSE, goes out as its own text, and the call raises rlmstudio_api_error with status 400. A reply text that breaks the id rules below, such as an empty text or a text of whitespace only, aborts before the request, as such an id does. The character vector that lms_chat_batch() returns with format = "vector" and the output column of its data frame carry no attribute. The data frame holds the ids in its response_id column.

The id, the attribute when there is one, must be valid in its declared encoding and not marked "bytes". A class, names, and the S4 bit are removed before the id is sent. NULL, the default, starts a new thread. It needs api_type = "native" or api_type = "openresponses". With api_type = "openai", an id or a value that carries one aborts before the request, because the OpenAI chat endpoint keeps no thread. As the id, NA, an empty string, a string of whitespace only, a value that is not a string, and more or fewer than one string abort before the check for a running server. For an attribute, the message names the attribute. Only this exact argument name is checked. A shortened name, such as previous, goes into the request body unchecked, under the name you wrote. An id that the server does not hold raises rlmstudio_api_error with status 400.

store

TRUE, FALSE, or NULL. Whether the server stores the reply, so that a later call can continue from its id. NULL, the default, sends no store field, so the server default applies, and that default stores the reply. TRUE and FALSE go out as a plain JSON true or false, also when the value has names, dimensions, or a class. Any other value, NA included, aborts before the check for a running server. A TRUE or FALSE needs api_type = "native" or api_type = "openresponses". NULL passes on every route. With api_type = "openai", TRUE or FALSE aborts before the request, because the OpenAI chat endpoint keeps no thread. Only this exact name is checked. A shortened name, such as sto, goes into the request body unchecked, under the name you wrote.

token

Character or NULL. An API token for a server that requires authentication. NULL reads the rlmstudio.token option and then the RLMSTUDIO_API_TOKEN environment variable. See rlmstudio_token.

Details

This function calls lms_chat_openresponses(), lms_chat_openai(), or lms_chat_native(), according to api_type. It runs no request of its own. It can raise rlmstudio_no_server and rlmstudio_api_error through lms_chat_openresponses(), lms_chat_openai(), or lms_chat_native().

With simplify = TRUE, it can raise rlmstudio_bad_response through any of the three, for a reply that holds no readable answer text. With either setting of simplify, it raises rlmstudio_bad_response through any of the three for a status-200 body that does not parse as JSON. With either setting of simplify, it raises rlmstudio_model_mismatch through lms_chat_openresponses() or lms_chat_openai() for a reply from a model other than the one asked for.

Value

Depending on the arguments provided:

- If simplify = FALSE, returns a parsed list of the raw JSON response.
- If simplify = TRUE and logprobs = FALSE, returns a single character string containing the model's text response. With a schema, it returns the reply parsed into an R value instead.
- If simplify = TRUE and logprobs = TRUE on the "openresponses" route, returns an object of class lms_chat_result containing both the text and a data.frame of token probabilities, or the text alone when no part of the reply carries log probabilities. With api_type = "openai", the logprobs of that object is NULL, and the call gives a warning of class rlmstudio_no_logprobs. The rlmstudio.quiet option does not hide it.

With simplify = TRUE and api_type = "native" or api_type = "openresponses", the value carries the id of the reply in a response_id attribute. Pass the value itself as previous_response_id to continue the thread, and the attribute is sent. The id is the response_id field of a native reply and the id field of an OpenResponses reply. The value has no attribute when that field is absent or is not one string. With api_type = "openai", or with simplify = FALSE, the value has no response_id attribute.

A native reply sent with store = FALSE has no response_id field, so its value has no attribute. An OpenResponses reply sent with store = FALSE still has an id, so its value carries the attribute, but the server does not hold that reply. A later call on the OpenResponses route that passes that id as previous_response_id raises rlmstudio_api_error with status 400 and the code "previous_response_not_found".

Server not running

Functions that call the LM Studio REST API open a TCP connection to the hostname and port named in host before they send the request. A function that checks its own arguments does that first, so a bad model, job_id, input, inputs, messages, schema, ttl, previous_response_id, or batch_size, a bad type of list_models() or list_instances(), a bad TRUE or FALSE argument such as simplify, logprobs, quiet, force, or store, a store on the "openai" route of lms_chat() or lms_chat_batch(), a stream in the ... of a chat function, a logprobs in the ... of lms_chat_batch() or lms_chat_native(), a value in the ... of lms_chat_batch() that lms_chat() reads as an argument already given, an instructions or messages in the ... of lms_chat() or lms_chat_batch() on the route where lms_chat() sets it, or a name, id, or type string that is not valid in its encoding or is marked as bytes, aborts with an argument message and no condition class even when the server is down. A condition of class rlmstudio_no_server is raised when that connection cannot be opened. A refused connection raises it. So do an address the package cannot parse and a hostname that does not resolve. An address that neither accepts nor refuses the connection also raises it. That case waits for the operating system to give up, which can take a minute. If the hostname of host is localhost, in any letter case, or 127.0.0.1, start the server with lms_server_start() and the port of host. For the default host, that is lms_server_start(port

= 1234). With no port, LM Studio uses the port that it used last. For any other host, start the server on the machine that host names, or give host the address that your server listens on. A host that does not parse gets this second hint too. The condition message gives the same hint for the host of the call.

The check reads the port and nothing else. Any process holding that port accepts the connection, so the condition is not raised even though no LM Studio server is there. The call then does not raise rlmstudio_no_server, and what it does depends on what answers. On a status-200 body that does not parse as JSON, the chat functions, lms_embed(), list_models(), list_instances(), lms_load(), lms_download(), lms_download_status(), and lms_unload_all() raise rlmstudio_bad_response. lms_unload() does not read the body, so it can report success. A body that parses as JSON but has another shape can come back unchanged with simplify = FALSE. With simplify = TRUE, the chat functions and lms_embed() raise rlmstudio_bad_response for it. list_models() and list_instances() raise it for a model list with another shape, and so do lms_unload_all() and lms_load() without force = TRUE, which read that list. lms_chat_openai() and lms_chat_openresponses() raise it for such a model list too, with either setting of simplify, through the model lookup that a reply from another model starts. lms_chat() raises it through them. lms_embed() raises it in the same way for the inputs of that request. lms_load(), lms_download(), and lms_download_status() raise it for a reply of their own with another shape, such as {}. A process that does not answer in HTTP gives an httr2_failure error. Use lms_server_ready() for the stronger test: it asks the host for a model list and reports TRUE only for a model list that list_models() can read.

lms_chat_batch() checks the server once before its first input, and lms_chat() checks it again for each input. If that check finds the server gone during the batch, the batch aborts with rlmstudio_no_server, and no request goes out after that. The condition then carries a results field, a list as long as inputs. Its elements before the lost input hold the values that format = "list" returns for those inputs. The element of the lost input and every element after it are NULL. The check before the first input adds no results field. A connection that fails after the check passes, such as a server that stops during a request, raises an httr2_failure error instead. That error aborts the batch and carries no results field.

lms_embed() checks the server before each request, and a request carries at most batch_size inputs. If a check after the first request, or the check of a model-list lookup, finds the server gone, the call aborts with rlmstudio_no_server. Once a request has succeeded, the condition carries a results field. With simplify = TRUE, results is a matrix with NA in each row whose embedding did not arrive. With simplify = FALSE, it is a list with one element per batch, with NULL in the element of the request that ended the call and in every element after it.

API failure

A condition of class rlmstudio_api_error is raised when a REST call returns a response that the wrapper treats as a failure. The condition carries a status field, which holds the HTTP response status as an integer. It also carries a code field. The field holds the string at error.code of the response body, such as "model_not_found". It is NULL when the body does not parse, when error is not a JSON object, or when its code is not one string. A server_message field holds the error text as the server sent it: the error string of the body, or else the string at error.message. It is NULL when the body does not parse or holds neither string.

On LM Studio 0.4.26+4 with no model loaded, /v1/chat/completions, /v1/responses, and /v1/embeddings answered a model name that the server could not find with status 400, no code, and a server_message that starts with "No models loaded".

`lms_chat_batch()` aborts on it when its `status` is 401, 403, or 404, when its `status` is 400 and its `code` is `"model_not_found"`, and when its `status` is 400 and its `server_message` starts with "No models loaded", in that letter case. No request goes out after that input. The condition then carries a `results` field that follows the rule for a lost server in the "Server not running" section: its elements before the failed input hold the values that `format = "list"` returns for those inputs, and the element of the failed input and every element after it are `NULL`. For any other status, the element of the failed input holds the condition, or `NA` where the result is text, and the batch warns once and goes on. See the details of `lms_chat_batch()`.

`lms_embed()` follows the same rule for each request, except for the code `"model_not_found"`. A 401, 403, or 404 aborts the call, and so does a 400 whose `server_message` starts with "No models loaded". Once a request has succeeded, the condition carries a `results` field, a matrix or a list as the "Server not running" section describes. Any other status, a 400 with the code `"model_not_found"` included, fails the inputs of that request alone, and the call warns once and goes on. If every request fails, the call aborts with the first condition. See the details of `lms_embed()`.

Malformed response

A condition of class `rlmstudio_bad_response` is raised when the server answers with a status the wrapper accepts and a body the wrapper cannot read. It is raised where a wrapper checks the body before it reshapes it, rather than indexing straight into whatever arrived. Eleven functions raise it for a body they cannot read: `lms_embed()`, `lms_chat()`, `lms_chat_native()`, `lms_chat_openresponses()`, `lms_chat_openai()`, `list_models()`, `list_instances()`, `lms_load()`, `lms_download()`, `lms_download_status()`, and `lms_unload_all()`. `lms_chat()` raises it through the chat function it calls. `lms_unload_all()` raises it through `list_models()`, and so does `lms_load()` unless `force = TRUE`. `lms_chat_batch()` raises it only as `rlmstudio_model_mismatch`, which the "Reply from another model" section of `lms_chat_openai()` describes.

All eleven raise it for a status-200 body that does not parse as JSON, such as an HTML page from a proxy, JSON text that stops part way, or an empty body. In the functions that take `simplify`, the body is parsed before `simplify` is read, so the condition is raised whatever `simplify` is. The body is parsed by its content and not by its `Content-Type` header, so valid JSON under `text/plain` is read as JSON. The body is read as JSON text and nothing else. A body whose text is a URL or the path of a file does not parse, and the package does not fetch the URL or read the file. The message says that the body did not parse as JSON and that something other than LM Studio may be answering on the host. It does not hold the body text.

The condition carries a `status` field, which holds the HTTP response status as an integer. Today the status is always 200: each of these functions reads the body only after a 200, and reports every other status as an `rlmstudio_api_error` instead.

Malformed model list

`list_models()` also raises it for a status-200 model list with the wrong shape. `lms_unload_all()` and `lms_load()` without `force = TRUE` raise it through `list_models()`. `lms_chat_openai()`, `lms_chat_openresponses()`, and `lms_embed()` apply these rules to the model list of their model lookup, as the "Reply from another model" section of `lms_chat_openai()` describes. A model list must follow four rules. Each field is read by its exact name, so a field named `keyX` does not stand in for `key`.

1. The body is a JSON object whose `models` field is an array. The array can be empty.

2. Each entry of `models` is a JSON object. Its `type` and `key` are strings, and its `loaded_instances` is an array.
3. The `size_bytes` of an entry is a number, or absent, or `null`. So is its `max_context_length`.
4. Each entry of `loaded_instances` is a JSON object whose `id` is a string with a character that is not whitespace.

The rules are checked before the `type` and `loaded` filters, so an entry that the filters drop can still raise the condition. The message names the field or entry that broke a rule. `lms_server_ready()` applies the same rules and returns `FALSE` for a body that breaks one.

`list_instances()` applies the four rules too, and three more. They cover only an entry whose `type` is in its `type` argument and that has at least one loaded instance. The `display_name` of such an entry is a string, or absent, or `null`. The `remaining_ttl_seconds` of each of its instances is a number, or absent, or `null`. The `config` of each of its instances is a JSON object, or absent, or `null`. `list_models()` and `lms_server_ready()` do not apply these three rules.

Malformed chat reply

`lms_chat_native()` and `lms_chat_openresponses()` raise it with `simplify = TRUE` when the reply holds no readable answer text. Both read the answer from the items of type `"message"` in the `output` array. They raise it when `output` is missing, empty, or not an array, or when an item in it is not a JSON object. They also raise it when no item has the type `"message"`, as in a reply that holds only reasoning or a tool call. The text of a message must be one string. For `lms_chat_native()` that is the `content` of the item. For `lms_chat_openresponses()` it is the `text` of each part of type `"output_text"`. For `lms_chat_openresponses()` only, the `content` of each message must be an array of JSON objects, and the messages together must hold at least one `"output_text"` part.

Apart from a body that does not parse as JSON, `lms_chat_openai()` raises it in four cases, all only with `simplify = TRUE`. The first case is a response whose `choices` field is missing, empty, or not an array, or whose first element is not a JSON object with a `message` object in it, so there is no reply to read. This case is raised with or without a `schema`, and with `logprobs = TRUE` as well. The second case is a reply that does not parse. A `schema` was given, `logprobs = FALSE`, and the reply content is not one string of valid JSON. The third case is reply content that is not one string, such as `null`, a missing `content` field, a number, or an array. A reply that holds only a tool call has `null` content. This case is raised without a `schema`, and with `logprobs = TRUE` with or without one. The fourth case is a cut-off reply. A `schema` was given, `logprobs = FALSE`, and the server reports the finish reason `"length"`. This case is raised also when the reply content parses, because a reply that stops part way can parse to a wrong value, such as the first digit of a longer number. In the second, third, and fourth cases, if the server reports the finish reason `"length"`, a length limit ended the reply. The limit is `max_tokens` or the context length of the model. The message then says so and names both.

With `simplify = TRUE`, `lms_chat_native()`, `lms_chat_openresponses()`, and `lms_chat_openai()` also raise it for a body that is a bare JSON value, such as `5`, `"s"`, or `true`. The message says that the response body is not a JSON object. A body of `null` gets the message about its missing `output` or `choices` field instead. `lms_chat()` can raise the condition through all three chat functions.

`lms_chat_batch()` does not abort on it, except on the subclass `rlmstudio_model_mismatch`, as the "Reply from another model" section of `lms_chat_openai()` says. The element of the failed input holds the condition, or `NA` where the result is text, and the batch warns once and goes on. See the details of `lms_chat_batch()`.

A condition from lms_chat_openai() about its reply also carries two more fields. A condition from the model-list lookup does not, as the "Reply from another model" section of lms_chat_openai() says. The content field holds the reply content of the first choice, and the finish_reason field holds the finish reason of the first choice. Both are NULL for a response with no choices. In the third case, content holds the value that was read, which is NULL for null or missing content. For the second, third, and fourth cases, the message names the content field, so you can read what the model wrote without a second request. The other messages of the chat functions about the reply name simplify = FALSE, which returns the body unchanged, with one exception. A body that did not parse as JSON is checked before that argument is read, so its message points at the host instead. For such a body, the content and finish_reason fields of a condition from lms_chat_openai() are NULL. The messages of rlmstudio_model_mismatch and of the model-list lookup do not name simplify = FALSE, because the check runs with either setting of simplify.

Malformed logprobs

With simplify = TRUE and logprobs = TRUE, lms_chat_openresponses() also raises it for a logprobs value that breaks one of these rules. The logprobs value of each "output_text" part is checked. A logprobs value, token, logprob, or top_logprobs that is null or absent passes its rule. A null step or candidate breaks rule 2 or rule 6.

1. The value is an array.
2. Each step in the array is a JSON object.
3. The token of a step is a string.
4. The logprob of a step is a number.
5. The top_logprobs of a step is an array.
6. Each candidate in top_logprobs is a JSON object.
7. The token and logprob of each candidate follow rules 3 and 4.

The parts are checked in order, then the steps of a part, then the candidates of a step, one at a time. Within a step, rules 3, 4, and 5 come before the candidates. The message names the first broken rule that this order reaches. These checks run only after the text of every "output_text" part is read, so a reply that also has a bad text in any part gets the text message. Parts of other types, such as a refusal, are not checked, and with logprobs = FALSE no part is checked. Fields are read by their exact names, so a field whose name only starts with the one asked for, such as tokenX, reads as absent and gives NA in the data frame.

Cut-off reply

A warning of class rlmstudio_reply_cut_off is given when a length limit ended a reply that lms_chat_openai() returns as text. The server reports this with the finish reason "length". The limit is max_tokens or the context length of the model, and the reply does not say which one. The message names both. The warning is given with simplify = TRUE for reply content that is one string, in three settings: no schema, logprobs = TRUE, and a schema with logprobs = TRUE. The call returns what the same reply returns without the cut-off. A schema reply with logprobs = FALSE raises rlmstudio_bad_response instead, as the "Malformed chat reply" section says. The warning and that abort read the finish reason of the first choice, which is the choice that the call reads. No other element of choices is read. With simplify = FALSE, the call returns the body with every choice and no warning.

lms_chat() gives the warning through lms_chat_openai() with api_type = "openai". lms_chat_batch() gives one warning of this class for the whole batch in place of one for each input. It names the count and the positions of the cut-off inputs, and each of those elements keeps its reply. It comes after the warning about failed inputs. A batch that aborts with a results field gives this warning before the abort. It names the cut-off inputs whose replies the results field of the abort holds. The native and OpenResponses routes give no such warning.

The warning shows whatever quiet and the rlmstudio.quiet option say, because it is the only sign that an answer is not complete.

Reply from another model

lms_chat_openai(), lms_chat_openresponses(), and lms_embed() read the model field of each status-200 reply, with either setting of simplify. On LM Studio 0.4.25+1 with one chat model loaded, both chat routes answered a model name that the server could not find with status 200 and a reply from the loaded model. With two chat models loaded, they answered with status 400 and the code "model_not_found", as the "API failure" section describes. On LM Studio 0.4.26+4 with one embedding model loaded, /v1/embeddings answered a name that it could not find with status 200 and vectors from the loaded embedding model. The model field names the loaded instance that answered. Its id can differ from the key that the call asked for.

If the model field equals the name that the call sent, the reply is accepted. If it differs, the call sends one request for the model list to api/v1/models on the same host, with the token of the call, and prints no message. The reply is accepted if a model whose key equals the asked name in any letter case has a loaded instance whose id is the model field. Otherwise the call aborts with a condition of class rlmstudio_model_mismatch. That condition is also an rlmstudio_bad_response, and its status is 200. Its model field holds the asked name, and its reply_model field holds the model field of the reply. The message names both. A condition from lms_chat_openai() also carries the content and finish_reason fields, both NULL.

Within one call of lms_chat_batch() or lms_embed(), a reply whose model field a lookup of that call already accepted for the asked name and the same host is accepted with no other request. Each such call, and each call of lms_chat_openai() or lms_chat_openresponses() outside a batch, starts with no accepted ids.

The check runs before the reply is read, so it also runs with logprobs = TRUE, with a schema on lms_chat_openai(), and for a reply with no answer text. In lms_embed(), it runs before the vectors are read. A reply is not checked if its body is not a JSON object, if it has no model field, or if its model is not one string or holds only whitespace. A body that is not a JSON object is returned with simplify = FALSE, and with simplify = TRUE it raises the error that the "Malformed chat reply" section describes, or for lms_embed() the "Malformed embeddings" section. If the instance that answered is unloaded before the model-list request, the call aborts, also when that instance belongs to the asked model.

If the model-list request fails, the check raises the condition of that failure. The message opens with the label of the function, followed by "because the model-list lookup failed". A status other than 200 raises rlmstudio_api_error. A body that does not parse as JSON, or that breaks a rule of a model list in the "Malformed model list" section, raises rlmstudio_bad_response. A server that the port check before the request cannot reach raises rlmstudio_no_server. A condition from the lookup does not carry the content and finish_reason fields, also when lms_chat_openai() raises it.

lms_chat() runs the check through lms_chat_openai() and lms_chat_openresponses(). lms_chat_native() does not check the reply. On LM Studio 0.4.25+1, /api/v1/chat answered a name that it could not find with status 404, which raises rlmstudio_api_error.

lms_chat_batch() aborts at the first input that raises rlmstudio_model_mismatch, on the "openai" and "openresponses" routes. It also aborts at an rlmstudio_api_error with status 400 whose code is "model_not_found", and at one with status 400 whose server_message starts with "No models loaded". No request goes out after that input. In each case, the condition carries a results field that follows the rule for a lost server in the "Server not running" section.

lms_embed() aborts at the first request that raises rlmstudio_model_mismatch, and no request goes out after it. A failed model-list lookup that raises rlmstudio_no_server, or an rlmstudio_api_error that would end the call as an embeddings failure, such as one with status 401, 403, or 404, ends the call in the same way. Once a request has succeeded, the condition carries a results field, a matrix or a list as the "Server not running" section describes. Any other rlmstudio_api_error or rlmstudio_bad_response from the lookup fails the inputs of that request alone, as the "API failure" section describes for a failed request. An error of any other class, such as an httr2_failure from a server that stops during the lookup, aborts the call unchanged.

Long prompts

A prompt longer than the context length of the loaded model fails. On LM Studio 0.4.25+1, with google/gemma-3-1b loaded at a context_length of 512, the /v1/responses and /api/v1/chat routes answered a longer prompt with status 500. The /v1/chat/completions route answered it with status 400. Each message began "The number of tokens to keep from the initial prompt is greater than the context length". The chat functions raise such a reply as an rlmstudio_api_error, and the message holds the server text. lms_chat_batch() fails that input alone and goes on to the next input. With format = "list", the element of that input holds the condition. With format = "vector", it holds NA.

To fit a longer prompt, load the model with a larger context_length in lms_load(). When the server reports it, the max_context_length column of list_models(detailed = TRUE) gives the largest context length that the model list reports for each model. If context_length is larger, the server loads the model with the asked value and gives no message. On LM Studio 0.4.25+1, it loaded google/gemma-3-1b at 65536 tokens, above its maximum of 32768. So lms_load() gives a warning of class rlmstudio_context_above_max that names both numbers, and then sends the load. The rlmstudio.quiet option does not hide it. The warning needs the model list, so it comes only with force = FALSE, and only when the list has a maximum for the model and the model is not loaded yet.

On LM Studio 0.4.25+1, the load endpoint answered a rope_frequency_scale field in ... with status 400 and the code "unrecognized_keys". So lms_load() cannot set RoPE scaling through that field. RoPE scaling stretches the position encoding of a model past its trained length.

To see how many tokens a prompt took, use lms_chat_batch(format = "data.frame"). Its input_tokens column holds the prompt token count that the server reports for each reply, on every route.

`lms_chat_batch` *Batch Chat Completion with LM Studio*

Description

Process a vector of inputs sequentially through LM Studio.

Usage

```
lms_chat_batch(
  model,
  inputs,
  system_prompt = NULL,
  format = c("vector", "list", "data.frame"),
  host = "http://localhost:1234",
  simplify = TRUE,
  quiet = NULL,
  ...,
  token = NULL
)
```

Arguments

`model`	Character. The loaded model name. Must be one name, given as a single string. The string must be valid in its declared encoding and not marked `"bytes"`. A class, names, and the S4 bit are removed before the name is sent.
`inputs`	Character vector. The prompts to process. Must hold at least one value and no missing values.
`system_prompt`	Character. Optional system prompt.
`format`	Character. Output format: "vector", "list", or "data.frame".
`host`	Character. Server URL.
`simplify`	`TRUE` or `FALSE`. If `TRUE`, parses outputs. Any other value, `NULL` and `NA` included, aborts before the check for a running server.
`quiet`	`TRUE`, `FALSE`, or `NULL`, the default. `NULL` follows the `rlmstudio.quiet` option. `TRUE` starts no progress bar, and `FALSE` starts one, also when the option is `TRUE`. cli draws a started bar only after a delay, two seconds by default. Any other value, `NA` included, aborts before the check for a running server. `quiet` does not hide the warnings about failed inputs, cut-off replies, a vector format that returns a list, or a `logprobs = TRUE` that the `"native"` route ignores.
`...`	Additional arguments passed to `lms_chat`, such as `api_type`, `logprobs`, `schema`, `ttl`, `previous_response_id`, or `store`. A `schema`, a `ttl`, a `previous_response_id`, a `store`, and the `api_type` that each needs are checked before the first call. A `previous_response_id` goes to every call, so each input continues the same stored reply. It can be an earlier reply of `lms_chat()` itself, and the `response_id` attribute of that reply is sent, by the rules of `lms_chat()`. A reply that came

back with no id goes out as its own text, and each input then fails with status 400, unless that text breaks the id rules of lms_chat() and aborts first. The character vector of format = "vector" and the output column of format = "data.frame" carry no attribute, as described below. A store here goes to lms_chat() for every call, and it is checked there by the rules of lms_chat(). It must be TRUE, FALSE, or NULL, and NULL, the default, sends no store field. Any other value, NA included, aborts before the check for a running server. With api_type = "openai", TRUE or FALSE aborts before the check for a running server. Only this exact name is checked. A shortened name, such as sto, goes into each request body unchecked, under the name you wrote. With store = FALSE, a native reply has no response_id field. An OpenResponses reply still has an id, but the server does not hold that reply, so a later call on the OpenResponses route that passes it as previous_response_id gets status 400 with the code "previous_response_not_found". A logprobs here, or a shortened name that lms_chat() reads as logprobs, must be TRUE or FALSE. Any other value, NULL and NA included, aborts before the check for a running server. With api_type = "native", which has no logprobs, a logprobs of TRUE is ignored. The batch then returns what it returns with logprobs = FALSE, in each format. It gives one warning for the whole batch, after the check for a running server, even with quiet = TRUE. With api_type = "openai" and simplify = TRUE, the batch sends the flag, and the logprobs of each reply is NULL, because the package reads no log probabilities from that route. The batch then gives one warning of class rlmstudio_no_logprobs that counts those replies, after the failed-inputs and cut-off warnings. A batch that aborts on a lost server, a refused token, a model the server cannot find, or a reply from another model gives it before the abort. The rlmstudio.quiet option and quiet = TRUE do not hide it. A batch in which no reply is read, for example because every input fails, gives no such warning. With simplify = TRUE and logprobs = TRUE on the "openresponses" or "openai" route, format = "vector" returns the list and says why in a warning. The rlmstudio.quiet option and quiet = TRUE do not hide it. In a batch that returns, that reason is a line of the failed-inputs warning if an input failed. Otherwise it is its own warning, before any cut-off or no-logprobs warning. Two values here that lms_chat() reads as the same argument abort with a message that names the argument. Examples are logprobs = TRUE, logprobs = FALSE, the shortened names log = TRUE, lo = FALSE, and two previous_response_id values. An exact name and a shortened name, such as logprobs and log, do not abort, because R gives the shortened one to the ... of lms_chat(). An input here also aborts, because the batch passes each element of inputs as input. An input reaches ... only when inputs is given by its full name. Otherwise R reads input as a shortened inputs, so it becomes inputs, and the value given by position fills the next unnamed argument, such as system_prompt. These aborts come before every other check of ... and before the check for a running server. An instructions here on the "openresponses" route, the default, and a messages here on the "openai" route abort before the check for a running server, with the message that lms_chat() gives. A previous_response_id here follows the rules of lms_chat(). It must be valid in its declared encoding and not marked "bytes". A class, names, and the S4 bit are removed before

the id is sent. The package checks a stream here. A stream other than FALSE or NULL aborts before the call checks for a running server, because the package reads a whole reply and not a streamed one.

token — Character or NULL. An API token for a server that requires authentication. NULL reads the rlmstudio.token option and then the RLMSTUDIO_API_TOKEN environment variable. See rlmstudio_token.

Details

This function calls lms_chat() once for each element of inputs. It raises rlmstudio_no_server itself, before the first call.

An rlmstudio_bad_response that lms_chat() raises for one input fails that input alone, unless it is an rlmstudio_model_mismatch. So does an rlmstudio_api_error with any status other than 401, 403, or 404, unless its status is 400 and its code is "model_not_found" or its server_message starts with "No models loaded". The batch goes on to the next input. Where the result is a list, or the output list-column that a schema gives, the element for that input holds the condition without its backtrace. An rlmstudio_bad_response for reply content that does not parse keeps that content in its content field. Where the result is text, the element holds NA. The result is text with format = "vector" when it returns a vector (simplify = TRUE, no schema, and logprobs = FALSE or the native route), and with a data frame whose replies are not parsed (no schema, or logprobs = TRUE). On the OpenResponses and OpenAI routes, the logprobs column holds NULL for a failed input. A reply with no readable answer text, such as one whose content is null, fails as an rlmstudio_bad_response in the same way. So does a status-200 body that does not parse as JSON, such as an HTML page from a proxy or an empty body. That input fails alone, and the other elements keep their replies. Use format = "list" to keep the conditions. The call gives one warning that names the count and the positions of the failed inputs. That warning shows even with quiet = TRUE.

An rlmstudio_api_error with status 401, 403, or 404 aborts the batch with that condition, and no request goes out after that input. Such a status comes from a fault that does not depend on the prompt, such as a token that the server refuses, so every later input fails in the same way. The condition carries a results field, as described in the "API failure" section below. No warning about failed inputs is given.

The batch aborts in the same way at an rlmstudio_api_error with status 400 and the code "model_not_found", at one with status 400 whose server_message starts with "No models loaded", and at an rlmstudio_model_mismatch. The "openai" and "openresponses" routes give these for a model name that the server cannot find, so every later input fails in the same way. They gave the "No models loaded" status 400 when no model was loaded. Within one batch, a reply id that the model-list lookup accepted is not looked up again. See the "API failure" and "Reply from another model" sections below.

A previous_response_id that the server does not hold gives status 400 with the code "invalid_value" on the native route and "previous_response_not_found" on the OpenResponses route. The batch does not abort at either one. Each such input fails alone, as described above, and the batch goes on to the next input.

An rlmstudio_no_server from lms_chat() also aborts the batch. Its results field holds the results so far, as described in the "Server not running" section below. An error of any other class aborts the batch unchanged.

Value

The return type depends on the `format` argument:

- `"vector"`: A character vector of responses, with `NA` for an input that failed. This format is only supported if `simplify = TRUE`, and `logprobs = FALSE` or `api_type = "native"`. With a `schema`, it warns and returns the list instead. With `logprobs = TRUE` on the `"openresponses"` or `"openai"` route, it also warns and returns the list.
- `"list"`: A list where each element is the response corresponding to the provided input, or the condition for an input that failed. With a `schema`, `simplify = TRUE`, and `logprobs = FALSE`, each element that did not fail is the parsed reply.
- `"data.frame"`: A data.frame containing `input` and `output` columns, with `NA` in `output` for an input that failed. If `logprobs = TRUE` on the `"openresponses"` or `"openai"` route, an additional list-column named `logprobs` is included, with `NULL` for an input that failed. The `"native"` route adds no `logprobs` column. With a `schema` and `logprobs = FALSE`, `output` is a list-column of parsed replies, with the condition in place of an input that failed. Columns read from each reply follow `output`, or `logprobs` when it is there, as described below. With an object `schema` and `logprobs = FALSE`, one column per schema property comes after them, as described below.

On the native and OpenResponses routes with `simplify = TRUE`, `lms_chat()` returns each reply that carries an id with a `response_id` attribute, as its help describes. With `format = "list"`, each reply keeps it. So does each element of the list that `format = "vector"` returns with `logprobs = TRUE` on the OpenResponses route. The character vector of `format = "vector"` and the `output` column of `format = "data.frame"` carry no `response_id` attribute. The data frame holds the ids in its `response_id` column.

With `api_type = "native"` and `format = "data.frame"`, the data frame ends with seven columns read from each reply: `response_id`, `input_tokens`, `total_output_tokens`, `reasoning_output_tokens`, `tokens_per_second`, `time_to_first_token_seconds`, and `model_load_time_seconds`. `response_id` is character, and it identifies the reply on the server. The other six are double, and they come from the `stats` object of the reply. A cell is `NA` when its field is absent or is not one value of the column type, a string for `response_id` and a number for the others. An empty string is a string, so an empty `response_id` is kept. If `stats` is absent or is not a JSON object, all six stats cells are `NA`. The server can leave a field out, such as `model_load_time_seconds`. Such a cell does not fail the input and gives no warning.

With `api_type = "openresponses"` or `api_type = "openai"` and `format = "data.frame"`, the data frame has four columns read from each reply after `output`, or after `logprobs` when it is there. They are `response_id`, `input_tokens`, `total_output_tokens`, and `reasoning_output_tokens`. These are the first four native column names, but the servers send the values under other names:

- `response_id` is the reply's `id` on both routes.
- `input_tokens` is `usage.input_tokens` on the OpenResponses route and `usage.prompt_tokens` on the OpenAI route.
- `total_output_tokens` is `usage.output_tokens` on the OpenResponses route and `usage.completion_tokens` on the OpenAI route.
- `reasoning_output_tokens` is `usage.output_tokens_details.reasoning_tokens` on the OpenResponses route and `usage.completion_tokens_details.reasoning_tokens` on the OpenAI route.

The columns are there for every setting of `logprobs` and `schema`. `response_id` is character, and the three counts are double. The `NA` rule is the one for the native columns: a cell is `NA` when its field is absent or is not one value of the column type, and an empty string is kept. If `usage` is absent or is not a JSON object, all three count cells are `NA`. If the details object is absent or is not a JSON object, only `reasoning_output_tokens` is `NA`. Such a cell does not fail the input and gives no warning.

A reply with no readable answer text fails its input, whatever its other fields hold. The row of an input that failed holds `NA` in every column read from the reply. If every input failed, those columns are still there, `response_id` as character and the others as double. The vector and list formats add no such column.

With `api_type = "openai"`, `format = "data.frame"`, `logprobs = FALSE`, and an object `schema`, the data frame ends with one column per top-level property of the schema, after the four reply columns. An object schema has a `type` of `"object"`, written as a string or as a list that holds that one string. Its `properties` is a list of one or more named entries. Each column has the property name, in the order of `properties`. The columns come from the schema and not from the replies, so they are there even when every input failed. The properties of a nested object get no columns of their own. Any other schema adds no column.

The type of a property column follows the `type` of the property. `"string"` gives character, `"integer"` gives integer, `"number"` gives double, and `"boolean"` gives logical. Each type can be a string or a list that holds that one string. A pair of one of these types and `"null"`, such as `c("integer", "null")`, gives the same column type. Any other `type`, no `type`, or a property that is not a list gives a list-column.

A cell of a character, integer, double, or logical property column holds the field of the parsed reply when the field is one value of the column type. A one-item JSON array counts as one value, and an empty string is kept. A double column takes any number. An integer column takes an integer, or a whole number from -2147483647 to 2147483647. Otherwise the cell is `NA`, and the call gives no warning. The `output` column keeps the parsed reply, with the value that did not fit. A cell of a list-column holds the field as parsed, or `NULL` when the field is absent or `null`. A reply that is not a JSON object, such as an array or a number, gives `NA` and `NULL` cells. So does the row of an input that failed.

A property name that is empty, `NA`, repeated, or equal to another column name aborts before the call checks for a running server. The other column names are `input`, `output`, `response_id`, `input_tokens`, `total_output_tokens`, and `reasoning_output_tokens`. A data frame with `logprobs = TRUE`, and the list and vector formats, do not check the names.

Server not running

Functions that call the LM Studio REST API open a TCP connection to the hostname and port named in `host` before they send the request. A function that checks its own arguments does that first, so a bad `model`, `job_id`, `input`, `inputs`, `messages`, `schema`, `ttl`, `previous_response_id`, or `batch_size`, a bad `type` of `list_models()` or `list_instances()`, a bad `TRUE` or `FALSE` argument such as `simplify`, `logprobs`, `quiet`, `force`, or `store`, a `store` on the `"openai"` route of `lms_chat()` or `lms_chat_batch()`, a `stream` in the `...` of a chat function, a `logprobs` in the `...` of `lms_chat_batch()` or `lms_chat_native()`, a value in the `...` of `lms_chat_batch()` that `lms_chat()` reads as an argument already given, an `instructions` or `messages` in the `...` of `lms_chat()` or `lms_chat_batch()` on the route where `lms_chat()` sets it, or a name, id, or `type` string that is not valid in its encoding or is marked as bytes, aborts with an argument message and no condition class even when the server is down. A condition of class `rlmstudio_no_server` is raised

when that connection cannot be opened. A refused connection raises it. So do an address the package cannot parse and a hostname that does not resolve. An address that neither accepts nor refuses the connection also raises it. That case waits for the operating system to give up, which can take a minute. If the hostname of `host` is `localhost`, in any letter case, or `127.0.0.1`, start the server with `lms_server_start()` and the port of `host`. For the default `host`, that is `lms_server_start(port = 1234)`. With no `port`, LM Studio uses the port that it used last. For any other `host`, start the server on the machine that `host` names, or give `host` the address that your server listens on. A `host` that does not parse gets this second hint too. The condition message gives the same hint for the `host` of the call.

The check reads the port and nothing else. Any process holding that port accepts the connection, so the condition is not raised even though no LM Studio server is there. The call then does not raise `rlmstudio_no_server`, and what it does depends on what answers. On a status-200 body that does not parse as JSON, the chat functions, `lms_embed()`, `list_models()`, `list_instances()`, `lms_load()`, `lms_download()`, `lms_download_status()`, and `lms_unload_all()` raise `rlmstudio_bad_response`. `lms_unload()` does not read the body, so it can report success. A body that parses as JSON but has another shape can come back unchanged with `simplify = FALSE`. With `simplify = TRUE`, the chat functions and `lms_embed()` raise `rlmstudio_bad_response` for it. `list_models()` and `list_instances()` raise it for a model list with another shape, and so do `lms_unload_all()` and `lms_load()` without `force = TRUE`, which read that list. `lms_chat_openai()` and `lms_chat_openresponses()` raise it for such a model list too, with either setting of `simplify`, through the model lookup that a reply from another model starts. `lms_chat()` raises it through them. `lms_embed()` raises it in the same way for the inputs of that request. `lms_load()`, `lms_download()`, and `lms_download_status()` raise it for a reply of their own with another shape, such as `{}`. A process that does not answer in HTTP gives an `httr2_failure` error. Use `lms_server_ready()` for the stronger test: it asks the host for a model list and reports `TRUE` only for a model list that `list_models()` can read.

`lms_chat_batch()` checks the server once before its first input, and `lms_chat()` checks it again for each input. If that check finds the server gone during the batch, the batch aborts with `rlmstudio_no_server`, and no request goes out after that. The condition then carries a `results` field, a list as long as `inputs`. Its elements before the lost input hold the values that `format = "list"` returns for those inputs. The element of the lost input and every element after it are `NULL`. The check before the first input adds no `results` field. A connection that fails after the check passes, such as a server that stops during a request, raises an `httr2_failure` error instead. That error aborts the batch and carries no `results` field.

`lms_embed()` checks the server before each request, and a request carries at most `batch_size` inputs. If a check after the first request, or the check of a model-list lookup, finds the server gone, the call aborts with `rlmstudio_no_server`. Once a request has succeeded, the condition carries a `results` field. With `simplify = TRUE`, `results` is a matrix with `NA` in each row whose embedding did not arrive. With `simplify = FALSE`, it is a list with one element per batch, with `NULL` in the element of the request that ended the call and in every element after it.

API failure

A condition of class `rlmstudio_api_error` is raised when a REST call returns a response that the wrapper treats as a failure. The condition carries a `status` field, which holds the HTTP response status as an integer. It also carries a `code` field. The field holds the string at `error.code` of the response body, such as `"model_not_found"`. It is `NULL` when the body does not parse, when `error` is not a JSON object, or when its `code` is not one string. A `server_message` field holds the error

text as the server sent it: the `error` string of the body, or else the string at `error.message`. It is `NULL` when the body does not parse or holds neither string.

On LM Studio 0.4.26+4 with no model loaded, `/v1/chat/completions`, `/v1/responses`, and `/v1/embeddings` answered a model name that the server could not find with status 400, no `code`, and a `server_message` that starts with "No models loaded".

`lms_chat_batch()` aborts on it when its `status` is 401, 403, or 404, when its `status` is 400 and its `code` is `"model_not_found"`, and when its `status` is 400 and its `server_message` starts with "No models loaded", in that letter case. No request goes out after that input. The condition then carries a `results` field that follows the rule for a lost server in the "Server not running" section: its elements before the failed input hold the values that `format = "list"` returns for those inputs, and the element of the failed input and every element after it are `NULL`. For any other status, the element of the failed input holds the condition, or `NA` where the result is text, and the batch warns once and goes on. See the details of `lms_chat_batch()`.

`lms_embed()` follows the same rule for each request, except for the code `"model_not_found"`. A 401, 403, or 404 aborts the call, and so does a 400 whose `server_message` starts with "No models loaded". Once a request has succeeded, the condition carries a `results` field, a matrix or a list as the "Server not running" section describes. Any other status, a 400 with the code `"model_not_found"` included, fails the inputs of that request alone, and the call warns once and goes on. If every request fails, the call aborts with the first condition. See the details of `lms_embed()`.

Malformed response

A condition of class `rlmstudio_bad_response` is raised when the server answers with a status the wrapper accepts and a body the wrapper cannot read. It is raised where a wrapper checks the body before it reshapes it, rather than indexing straight into whatever arrived. Eleven functions raise it for a body they cannot read: `lms_embed()`, `lms_chat()`, `lms_chat_native()`, `lms_chat_openresponses()`, `lms_chat_openai()`, `list_models()`, `list_instances()`, `lms_load()`, `lms_download()`, `lms_download_status()`, and `lms_unload_all()`. `lms_chat()` raises it through the chat function it calls. `lms_unload_all()` raises it through `list_models()`, and so does `lms_load()` unless `force = TRUE`. `lms_chat_batch()` raises it only as `rlmstudio_model_mismatch`, which the "Reply from another model" section of `lms_chat_openai()` describes.

All eleven raise it for a status-200 body that does not parse as JSON, such as an HTML page from a proxy, JSON text that stops part way, or an empty body. In the functions that take `simplify`, the body is parsed before `simplify` is read, so the condition is raised whatever `simplify` is. The body is parsed by its content and not by its `Content-Type` header, so valid JSON under `text/plain` is read as JSON. The body is read as JSON text and nothing else. A body whose text is a URL or the path of a file does not parse, and the package does not fetch the URL or read the file. The message says that the body did not parse as JSON and that something other than LM Studio may be answering on the host. It does not hold the body text.

The condition carries a `status` field, which holds the HTTP response status as an integer. Today the status is always 200: each of these functions reads the body only after a 200, and reports every other status as an `rlmstudio_api_error` instead.

Malformed model list

`list_models()` also raises it for a status-200 model list with the wrong shape. `lms_unload_all()` and `lms_load()` without `force = TRUE` raise it through `list_models()`. `lms_chat_openai()`,

lms_chat_openresponses(), and lms_embed() apply these rules to the model list of their model lookup, as the "Reply from another model" section of lms_chat_openai() describes. A model list must follow four rules. Each field is read by its exact name, so a field named keyX does not stand in for key.

1. The body is a JSON object whose models field is an array. The array can be empty.
2. Each entry of models is a JSON object. Its type and key are strings, and its loaded_instances is an array.
3. The size_bytes of an entry is a number, or absent, or null. So is its max_context_length.
4. Each entry of loaded_instances is a JSON object whose id is a string with a character that is not whitespace.

The rules are checked before the type and loaded filters, so an entry that the filters drop can still raise the condition. The message names the field or entry that broke a rule. lms_server_ready() applies the same rules and returns FALSE for a body that breaks one.

list_instances() applies the four rules too, and three more. They cover only an entry whose type is in its type argument and that has at least one loaded instance. The display_name of such an entry is a string, or absent, or null. The remaining_ttl_seconds of each of its instances is a number, or absent, or null. The config of each of its instances is a JSON object, or absent, or null. list_models() and lms_server_ready() do not apply these three rules.

Malformed chat reply

lms_chat_native() and lms_chat_openresponses() raise it with simplify = TRUE when the reply holds no readable answer text. Both read the answer from the items of type "message" in the output array. They raise it when output is missing, empty, or not an array, or when an item in it is not a JSON object. They also raise it when no item has the type "message", as in a reply that holds only reasoning or a tool call. The text of a message must be one string. For lms_chat_native() that is the content of the item. For lms_chat_openresponses() it is the text of each part of type "output_text". For lms_chat_openresponses() only, the content of each message must be an array of JSON objects, and the messages together must hold at least one "output_text" part.

Apart from a body that does not parse as JSON, lms_chat_openai() raises it in four cases, all only with simplify = TRUE. The first case is a response whose choices field is missing, empty, or not an array, or whose first element is not a JSON object with a message object in it, so there is no reply to read. This case is raised with or without a schema, and with logprobs = TRUE as well. The second case is a reply that does not parse. A schema was given, logprobs = FALSE, and the reply content is not one string of valid JSON. The third case is reply content that is not one string, such as null, a missing content field, a number, or an array. A reply that holds only a tool call has null content. This case is raised without a schema, and with logprobs = TRUE with or without one. The fourth case is a cut-off reply. A schema was given, logprobs = FALSE, and the server reports the finish reason "length". This case is raised also when the reply content parses, because a reply that stops part way can parse to a wrong value, such as the first digit of a longer number. In the second, third, and fourth cases, if the server reports the finish reason "length", a length limit ended the reply. The limit is max_tokens or the context length of the model. The message then says so and names both.

With simplify = TRUE, lms_chat_native(), lms_chat_openresponses(), and lms_chat_openai() also raise it for a body that is a bare JSON value, such as 5, "s", or true. The message says that

the response body is not a JSON object. A body of `null` gets the message about its missing `output` or `choices` field instead. `lms_chat()` can raise the condition through all three chat functions.

`lms_chat_batch()` does not abort on it, except on the subclass `rlmstudio_model_mismatch`, as the "Reply from another model" section of `lms_chat_openai()` says. The element of the failed input holds the condition, or `NA` where the result is text, and the batch warns once and goes on. See the details of `lms_chat_batch()`.

A condition from `lms_chat_openai()` about its reply also carries two more fields. A condition from the model-list lookup does not, as the "Reply from another model" section of `lms_chat_openai()` says. The `content` field holds the reply content of the first choice, and the `finish_reason` field holds the finish reason of the first choice. Both are `NULL` for a response with no `choices`. In the third case, `content` holds the value that was read, which is `NULL` for `null` or missing content. For the second, third, and fourth cases, the message names the `content` field, so you can read what the model wrote without a second request. The other messages of the chat functions about the reply name `simplify = FALSE`, which returns the body unchanged, with one exception. A body that did not parse as JSON is checked before that argument is read, so its message points at the host instead. For such a body, the `content` and `finish_reason` fields of a condition from `lms_chat_openai()` are `NULL`. The messages of `rlmstudio_model_mismatch` and of the model-list lookup do not name `simplify = FALSE`, because the check runs with either setting of `simplify`.

Malformed logprobs

With `simplify = TRUE` and `logprobs = TRUE`, `lms_chat_openresponses()` also raises it for a `logprobs` value that breaks one of these rules. The `logprobs` value of each `"output_text"` part is checked. A `logprobs` value, `token`, `logprob`, or `top_logprobs` that is `null` or absent passes its rule. A `null` step or candidate breaks rule 2 or rule 6.

1. The value is an array.
2. Each step in the array is a JSON object.
3. The `token` of a step is a string.
4. The `logprob` of a step is a number.
5. The `top_logprobs` of a step is an array.
6. Each candidate in `top_logprobs` is a JSON object.
7. The `token` and `logprob` of each candidate follow rules 3 and 4.

The parts are checked in order, then the steps of a part, then the candidates of a step, one at a time. Within a step, rules 3, 4, and 5 come before the candidates. The message names the first broken rule that this order reaches. These checks run only after the text of every `"output_text"` part is read, so a reply that also has a bad `text` in any part gets the text message. Parts of other types, such as a refusal, are not checked, and with `logprobs = FALSE` no part is checked. Fields are read by their exact names, so a field whose name only starts with the one asked for, such as `tokenX`, reads as absent and gives `NA` in the data frame.

Cut-off reply

A warning of class `rlmstudio_reply_cut_off` is given when a length limit ended a reply that `lms_chat_openai()` returns as text. The server reports this with the finish reason `"length"`. The limit is `max_tokens` or the context length of the model, and the reply does not say which one.

The message names both. The warning is given with `simplify = TRUE` for reply content that is one string, in three settings: no `schema`, `logprobs = TRUE`, and a `schema` with `logprobs = TRUE`. The call returns what the same reply returns without the cut-off. A `schema` reply with `logprobs = FALSE` raises `rlmstudio_bad_response` instead, as the "Malformed chat reply" section says. The warning and that abort read the finish reason of the first choice, which is the choice that the call reads. No other element of `choices` is read. With `simplify = FALSE`, the call returns the body with every choice and no warning.

`lms_chat()` gives the warning through `lms_chat_openai()` with `api_type = "openai"`. `lms_chat_batch()` gives one warning of this class for the whole batch in place of one for each input. It names the count and the positions of the cut-off inputs, and each of those elements keeps its reply. It comes after the warning about failed inputs. A batch that aborts with a `results` field gives this warning before the abort. It names the cut-off inputs whose replies the `results` field of the abort holds. The native and OpenResponses routes give no such warning.

The warning shows whatever `quiet` and the `rlmstudio.quiet` option say, because it is the only sign that an answer is not complete.

Reply from another model

`lms_chat_openai()`, `lms_chat_openresponses()`, and `lms_embed()` read the `model` field of each status-200 reply, with either setting of `simplify`. On LM Studio 0.4.25+1 with one chat model loaded, both chat routes answered a model name that the server could not find with status 200 and a reply from the loaded model. With two chat models loaded, they answered with status 400 and the code `"model_not_found"`, as the "API failure" section describes. On LM Studio 0.4.26+4 with one embedding model loaded, `/v1/embeddings` answered a name that it could not find with status 200 and vectors from the loaded embedding model. The `model` field names the loaded instance that answered. Its id can differ from the key that the call asked for.

If the `model` field equals the name that the call sent, the reply is accepted. If it differs, the call sends one request for the model list to `api/v1/models` on the same host, with the token of the call, and prints no message. The reply is accepted if a model whose key equals the asked name in any letter case has a loaded instance whose id is the `model` field. Otherwise the call aborts with a condition of class `rlmstudio_model_mismatch`. That condition is also an `rlmstudio_bad_response`, and its `status` is 200. Its `model` field holds the asked name, and its `reply_model` field holds the `model` field of the reply. The message names both. A condition from `lms_chat_openai()` also carries the `content` and `finish_reason` fields, both `NULL`.

Within one call of `lms_chat_batch()` or `lms_embed()`, a reply whose `model` field a lookup of that call already accepted for the asked name and the same host is accepted with no other request. Each such call, and each call of `lms_chat_openai()` or `lms_chat_openresponses()` outside a batch, starts with no accepted ids.

The check runs before the reply is read, so it also runs with `logprobs = TRUE`, with a `schema` on `lms_chat_openai()`, and for a reply with no answer text. In `lms_embed()`, it runs before the vectors are read. A reply is not checked if its body is not a JSON object, if it has no `model` field, or if its `model` is not one string or holds only whitespace. A body that is not a JSON object is returned with `simplify = FALSE`, and with `simplify = TRUE` it raises the error that the "Malformed chat reply" section describes, or for `lms_embed()` the "Malformed embeddings" section. If the instance that answered is unloaded before the model-list request, the call aborts, also when that instance belongs to the asked model.

If the model-list request fails, the check raises the condition of that failure. The message opens with the label of the function, followed by "because the model-list lookup failed". A status other than 200 raises `rlmstudio_api_error`. A body that does not parse as JSON, or that breaks a rule of a model list in the "Malformed model list" section, raises `rlmstudio_bad_response`. A server that the port check before the request cannot reach raises `rlmstudio_no_server`. A condition from the lookup does not carry the `content` and `finish_reason` fields, also when `lms_chat_openai()` raises it.

`lms_chat()` runs the check through `lms_chat_openai()` and `lms_chat_openresponses()`. `lms_chat_native()` does not check the reply. On LM Studio 0.4.25+1, `/api/v1/chat` answered a name that it could not find with status 404, which raises `rlmstudio_api_error`.

`lms_chat_batch()` aborts at the first input that raises `rlmstudio_model_mismatch`, on the `"openai"` and `"openresponses"` routes. It also aborts at an `rlmstudio_api_error` with status 400 whose `code` is `"model_not_found"`, and at one with status 400 whose `server_message` starts with "No models loaded". No request goes out after that input. In each case, the condition carries a `results` field that follows the rule for a lost server in the "Server not running" section.

`lms_embed()` aborts at the first request that raises `rlmstudio_model_mismatch`, and no request goes out after it. A failed model-list lookup that raises `rlmstudio_no_server`, or an `rlmstudio_api_error` that would end the call as an embeddings failure, such as one with status 401, 403, or 404, ends the call in the same way. Once a request has succeeded, the condition carries a `results` field, a matrix or a list as the "Server not running" section describes. Any other `rlmstudio_api_error` or `rlmstudio_bad_response` from the lookup fails the inputs of that request alone, as the "API failure" section describes for a failed request. An error of any other class, such as an `httr2_failure` from a server that stops during the lookup, aborts the call unchanged.

`lms_chat_native` *Chat Completion via Native API*

Description

Direct interface to LM Studio's v1 Native endpoint. Optimized for stateful chats and hardware control.

Usage

```
lms_chat_native(
  model,
  input,
  system_prompt = NULL,
  host = "http://localhost:1234",
  simplify = TRUE,
  ...,
  previous_response_id = NULL,
  store = NULL,
  token = NULL
)
```

Arguments

model	Character. The loaded model name. Must be one name, given as a single string. The string must be valid in its declared encoding and not marked "bytes". A class, names, and the S4 bit are removed before the name is sent.
input	Character. The user prompt, as one string. A character vector must hold no missing values, and its length must be one. Any other character value aborts before the check for a running server. To send several prompts, one request each, use lms_chat_batch(). A list is sent as given in the input field.
system_prompt	Character. Optional system prompt.
host	Character. Server URL.
simplify	TRUE or FALSE. If TRUE, parses output to text. Any other value, NULL and NA included, aborts before the check for a running server.
...	Additional API arguments. This endpoint rejects a ttl field with status 400, which raises rlmstudio_api_error. The package checks a stream here. A stream other than FALSE or NULL aborts before the call checks for a running server, because the package reads a whole reply and not a streamed one. The package also checks each element named exactly logprobs. It must be TRUE, FALSE, or NULL, and any other value aborts before the check for a running server. The endpoint has no logprobs, so no logprobs field goes into the request body, and a TRUE warns.
previous_response_id	One string, a value that carries a response_id attribute, or NULL. It names the stored reply that this chat continues. Pass an earlier reply of this function itself, such as first, or its id, attr(first, "response_id"). A value that carries a response_id attribute sends that attribute in place of the value, also when the value is a list or is itself an id. Only that exact attribute name is read. A value with no such attribute is sent as the id itself. So a reply that came back with no id, such as a reply of this function sent with store = FALSE, goes out as its own text, and the call raises rlmstudio_api_error with status 400. A reply text that breaks the id rules below, such as an empty text or a text of whitespace only, aborts before the request, as such an id does. The character vector that lms_chat_batch() returns with format = "vector" and the output column of its data frame carry no attribute. The data frame holds the ids in its response_id column. The id, the attribute when there is one, must be valid in its declared encoding and not marked "bytes". A class, names, and the S4 bit are removed before the id is sent. NULL, the default, starts a new thread. As the id, NA, an empty string, a string of whitespace only, a value that is not a string, and more or fewer than one string abort before the check for a running server. For an attribute, the message names the attribute. Only this exact argument name is checked. A shortened name, such as previous, goes into the request body unchecked, under the name you wrote. An id that the server does not hold raises rlmstudio_api_error with status 400 and the code "invalid_value". lms_chat() and lms_chat_batch() refuse an id or a value that carries one with api_type = "openai", because the OpenAI chat endpoint keeps no thread. lms_chat_openai() has no such argument. A previous_response_id in its ... goes into the request body unchecked, and the endpoint ignores it.

store — TRUE, FALSE, or NULL. Whether the server stores the reply, so that a later call can continue from its id. NULL, the default, sends no store field, so the server default applies, and that default stores the reply. TRUE and FALSE go out as a plain JSON true or false, also when the value has names, dimensions, or a class. Any other value, NA included, aborts before the check for a running server. Only this exact name is checked. A shortened name, such as sto, goes into the request body unchecked, under the name you wrote. lms_chat() and lms_chat_batch() refuse TRUE or FALSE with api_type = "openai".

token — Character or NULL. An API token for a server that requires authentication. NULL reads the rlmstudio.token option and then the RLMSTUDIO_API_TOKEN environment variable. See rlmstudio_token.

Value

If simplify = FALSE, returns a list representing the raw JSON response. A status-200 body that does not parse as JSON raises rlmstudio_bad_response with either setting of simplify. The body can hold a response_id for the reply and a stats object of token counts and timings. With api_type = "native" and format = "data.frame", lms_chat_batch() returns the id and six of the stats fields as columns. If simplify = TRUE, returns one character string: the content of every item of type "message" in the output array, pasted together in order with no separator. Items of other types, such as reasoning and tool calls, are skipped. A reply with no readable answer text raises rlmstudio_bad_response, as described below.

With simplify = TRUE, the string carries the response_id field of the reply in a response_id attribute. Pass the string itself as previous_response_id to continue the thread, and the attribute is sent. The string has no attribute when response_id is absent or is not one string. A reply sent with store = FALSE has no response_id field, so its string has no attribute, and there is no id to continue from. lms_chat_openresponses() reads the attribute from the id field of its reply instead. An OpenResponses reply sent with store = FALSE still has an id, so its value carries the attribute, but the server does not hold that reply. A later OpenResponses call that passes that id as previous_response_id raises rlmstudio_api_error with status 400 and the code "previous_response_not_found".

Server not running

Functions that call the LM Studio REST API open a TCP connection to the hostname and port named in host before they send the request. A function that checks its own arguments does that first, so a bad model, job_id, input, inputs, messages, schema, ttl, previous_response_id, or batch_size, a bad type of list_models() or list_instances(), a bad TRUE or FALSE argument such as simplify, logprobs, quiet, force, or store, a store on the "openai" route of lms_chat() or lms_chat_batch(), a stream in the ... of a chat function, a logprobs in the ... of lms_chat_batch() or lms_chat_native(), a value in the ... of lms_chat_batch() that lms_chat() reads as an argument already given, an instructions or messages in the ... of lms_chat() or lms_chat_batch() on the route where lms_chat() sets it, or a name, id, or type string that is not valid in its encoding or is marked as bytes, aborts with an argument message and no condition class even when the server is down. A condition of class rlmstudio_no_server is raised when that connection cannot be opened. A refused connection raises it. So do an address the package cannot parse and a hostname that does not resolve. An address that neither accepts nor refuses the connection also raises it. That case waits for the operating system to give up, which can take a

minute. If the hostname of host is localhost, in any letter case, or 127.0.0.1, start the server with lms_server_start() and the port of host. For the default host, that is lms_server_start(port = 1234). With no port, LM Studio uses the port that it used last. For any other host, start the server on the machine that host names, or give host the address that your server listens on. A host that does not parse gets this second hint too. The condition message gives the same hint for the host of the call.

The check reads the port and nothing else. Any process holding that port accepts the connection, so the condition is not raised even though no LM Studio server is there. The call then does not raise rlmstudio_no_server, and what it does depends on what answers. On a status-200 body that does not parse as JSON, the chat functions, lms_embed(), list_models(), list_instances(), lms_load(), lms_download(), lms_download_status(), and lms_unload_all() raise rlmstudio_bad_response. lms_unload() does not read the body, so it can report success. A body that parses as JSON but has another shape can come back unchanged with simplify = FALSE. With simplify = TRUE, the chat functions and lms_embed() raise rlmstudio_bad_response for it. list_models() and list_instances() raise it for a model list with another shape, and so do lms_unload_all() and lms_load() without force = TRUE, which read that list. lms_chat_openai() and lms_chat_openresponses() raise it for such a model list too, with either setting of simplify, through the model lookup that a reply from another model starts. lms_chat() raises it through them. lms_embed() raises it in the same way for the inputs of that request. lms_load(), lms_download(), and lms_download_status() raise it for a reply of their own with another shape, such as {}. A process that does not answer in HTTP gives an httr2_failure error. Use lms_server_ready() for the stronger test: it asks the host for a model list and reports TRUE only for a model list that list_models() can read.

lms_chat_batch() checks the server once before its first input, and lms_chat() checks it again for each input. If that check finds the server gone during the batch, the batch aborts with rlmstudio_no_server, and no request goes out after that. The condition then carries a results field, a list as long as inputs. Its elements before the lost input hold the values that format = "list" returns for those inputs. The element of the lost input and every element after it are NULL. The check before the first input adds no results field. A connection that fails after the check passes, such as a server that stops during a request, raises an httr2_failure error instead. That error aborts the batch and carries no results field.

lms_embed() checks the server before each request, and a request carries at most batch_size inputs. If a check after the first request, or the check of a model-list lookup, finds the server gone, the call aborts with rlmstudio_no_server. Once a request has succeeded, the condition carries a results field. With simplify = TRUE, results is a matrix with NA in each row whose embedding did not arrive. With simplify = FALSE, it is a list with one element per batch, with NULL in the element of the request that ended the call and in every element after it.

API failure

A condition of class rlmstudio_api_error is raised when a REST call returns a response that the wrapper treats as a failure. The condition carries a status field, which holds the HTTP response status as an integer. It also carries a code field. The field holds the string at error.code of the response body, such as "model_not_found". It is NULL when the body does not parse, when error is not a JSON object, or when its code is not one string. A server_message field holds the error text as the server sent it: the error string of the body, or else the string at error.message. It is NULL when the body does not parse or holds neither string.

On LM Studio 0.4.26+4 with no model loaded, /v1/chat/completions, /v1/responses, and

/v1/embeddings answered a model name that the server could not find with status 400, no code, and a server_message that starts with "No models loaded".

lms_chat_batch() aborts on it when its status is 401, 403, or 404, when its status is 400 and its code is "model_not_found", and when its status is 400 and its server_message starts with "No models loaded", in that letter case. No request goes out after that input. The condition then carries a results field that follows the rule for a lost server in the "Server not running" section: its elements before the failed input hold the values that format = "list" returns for those inputs, and the element of the failed input and every element after it are NULL. For any other status, the element of the failed input holds the condition, or NA where the result is text, and the batch warns once and goes on. See the details of lms_chat_batch().

lms_embed() follows the same rule for each request, except for the code "model_not_found". A 401, 403, or 404 aborts the call, and so does a 400 whose server_message starts with "No models loaded". Once a request has succeeded, the condition carries a results field, a matrix or a list as the "Server not running" section describes. Any other status, a 400 with the code "model_not_found" included, fails the inputs of that request alone, and the call warns once and goes on. If every request fails, the call aborts with the first condition. See the details of lms_embed().

Malformed response

A condition of class rlmstudio_bad_response is raised when the server answers with a status the wrapper accepts and a body the wrapper cannot read. It is raised where a wrapper checks the body before it reshapes it, rather than indexing straight into whatever arrived. Eleven functions raise it for a body they cannot read: lms_embed(), lms_chat(), lms_chat_native(), lms_chat_openresponses(), lms_chat_openai(), list_models(), list_instances(), lms_load(), lms_download(), lms_download_status(), and lms_unload_all(). lms_chat() raises it through the chat function it calls. lms_unload_all() raises it through list_models(), and so does lms_load() unless force = TRUE. lms_chat_batch() raises it only as rlmstudio_model_mismatch, which the "Reply from another model" section of lms_chat_openai() describes.

All eleven raise it for a status-200 body that does not parse as JSON, such as an HTML page from a proxy, JSON text that stops part way, or an empty body. In the functions that take simplify, the body is parsed before simplify is read, so the condition is raised whatever simplify is. The body is parsed by its content and not by its Content-Type header, so valid JSON under text/plain is read as JSON. The body is read as JSON text and nothing else. A body whose text is a URL or the path of a file does not parse, and the package does not fetch the URL or read the file. The message says that the body did not parse as JSON and that something other than LM Studio may be answering on the host. It does not hold the body text.

The condition carries a status field, which holds the HTTP response status as an integer. Today the status is always 200: each of these functions reads the body only after a 200, and reports every other status as an rlmstudio_api_error instead.

Malformed chat reply

lms_chat_native() and lms_chat_openresponses() raise it with simplify = TRUE when the reply holds no readable answer text. Both read the answer from the items of type "message" in the output array. They raise it when output is missing, empty, or not an array, or when an item in it is not a JSON object. They also raise it when no item has the type "message", as in a reply that holds only reasoning or a tool call. The text of a message must be one string. For lms_chat_native() that is the content of the item. For lms_chat_openresponses() it is the text of each part of type

"output_text". For lms_chat_openresponses() only, the content of each message must be an array of JSON objects, and the messages together must hold at least one "output_text" part.

Apart from a body that does not parse as JSON, lms_chat_openai() raises it in four cases, all only with simplify = TRUE. The first case is a response whose choices field is missing, empty, or not an array, or whose first element is not a JSON object with a message object in it, so there is no reply to read. This case is raised with or without a schema, and with logprobs = TRUE as well. The second case is a reply that does not parse. A schema was given, logprobs = FALSE, and the reply content is not one string of valid JSON. The third case is reply content that is not one string, such as null, a missing content field, a number, or an array. A reply that holds only a tool call has null content. This case is raised without a schema, and with logprobs = TRUE with or without one. The fourth case is a cut-off reply. A schema was given, logprobs = FALSE, and the server reports the finish reason "length". This case is raised also when the reply content parses, because a reply that stops part way can parse to a wrong value, such as the first digit of a longer number. In the second, third, and fourth cases, if the server reports the finish reason "length", a length limit ended the reply. The limit is max_tokens or the context length of the model. The message then says so and names both.

With simplify = TRUE, lms_chat_native(), lms_chat_openresponses(), and lms_chat_openai() also raise it for a body that is a bare JSON value, such as 5, "s", or true. The message says that the response body is not a JSON object. A body of null gets the message about its missing output or choices field instead. lms_chat() can raise the condition through all three chat functions.

lms_chat_batch() does not abort on it, except on the subclass rlmstudio_model_mismatch, as the "Reply from another model" section of lms_chat_openai() says. The element of the failed input holds the condition, or NA where the result is text, and the batch warns once and goes on. See the details of lms_chat_batch().

A condition from lms_chat_openai() about its reply also carries two more fields. A condition from the model-list lookup does not, as the "Reply from another model" section of lms_chat_openai() says. The content field holds the reply content of the first choice, and the finish_reason field holds the finish reason of the first choice. Both are NULL for a response with no choices. In the third case, content holds the value that was read, which is NULL for null or missing content. For the second, third, and fourth cases, the message names the content field, so you can read what the model wrote without a second request. The other messages of the chat functions about the reply name simplify = FALSE, which returns the body unchanged, with one exception. A body that did not parse as JSON is checked before that argument is read, so its message points at the host instead. For such a body, the content and finish_reason fields of a condition from lms_chat_openai() are NULL. The messages of rlmstudio_model_mismatch and of the model-list lookup do not name simplify = FALSE, because the check runs with either setting of simplify.

---

lms_chat_openai *Chat Completion via OpenAI Compatibility API*

Description

Direct interface to LM Studio's OpenAI-compatible endpoint. Uses the messages array format.

Usage

```
lms_chat_openai(
  model,
  messages,
  host = "http://localhost:1234",
  logprobs = FALSE,
  simplify = TRUE,
  ...,
  schema = NULL,
  ttl = NULL,
  token = NULL
)
```

Arguments

model	Character. The loaded model name. Must be one name, given as a single string. The string must be valid in its declared encoding and not marked "bytes". A class, names, and the S4 bit are removed before the name is sent.
messages	The messages to send. Give an unnamed list with one element per message, such as list(list(role = "user", content = "Hi")), or a data frame with at least one row. A data frame is sent as one message per row, with one field per column. A cell that is NA is left out of its message. The call aborts before it checks for a running server if messages breaks one of these rules:

- messages is a list or a data frame.
- A data frame has at least one row, and any other list has at least one element.
- A list that is not a data frame has no names. A single message not wrapped in a list, such as list(role = "user", content = "Hi"), breaks this rule.
- Each element of such a list is a list of length one or more, and each of its fields has a name that is not NA and not empty. A data frame as an element breaks this rule.
- A data frame has at least one column, and no column name is NA, empty, or repeated.
- No column of a data frame has a row count that differs from the row count of the data frame that holds it. jsonlite cannot write such a column. The rule reads each data-frame column, and each atomic or list column with no class or with the class "AsIs" alone. It reads such columns in the data frame and in its data-frame columns at any depth. The row count of a data-frame column is its own row count. The row count of a column with a dim attribute, such as a matrix, is the first extent of the dim. The row count of any other column is its length. So a character column of length one in a two-row data frame breaks this rule. The rule does not read a column with another class, such as a Date, a factor, or a POSIXlt time. jsonlite writes a POSIXlt column of length one with its one value in each message, and the rule below reads that value in each row. A Date or factor column of a wrong length breaks the last rule, unless an earlier rule refuses the data frame first.

- A data frame has no row in which every cell is NA or is a NULL cell of a list column. jsonlite leaves out an NA cell of another column and writes an NA or NULL list cell as null, so such a row holds no field value. A list() cell is sent as [] and a list(NA) cell as [null], so a row with such a cell is sent. A column with a dim attribute of any length whose first extent is the row count, such as a matrix or an array, counts as empty in a row when each of its cells in that row is empty. Such a column with no cells in a row, such as a 2-by-0 matrix, is not empty in that row, because jsonlite writes that row of it as [] or as nested empty arrays. A data-frame column counts as empty in a row when each of its own columns is empty in that row. So a data-frame column with no columns counts as empty in every row. A column that is neither an atomic vector nor a list, such as an environment, never counts as empty.
- messages, when it is a list and not a data frame, has no dim attribute, such as a matrix or an array of messages.
- A message that is a list has no dim attribute.
- No list inside a message has a dim attribute. The rule reads each field, each list at any depth below a field, and each cell of a list column. jsonlite would send such a list as nested arrays with each cell in an array of its own. A list column of a data frame with one, three, or more dimensions also breaks this rule. An atomic matrix field is sent as an array of arrays. A list-matrix column of any data frame is sent, one row of cells for each message. jsonlite does not unbox a value inside a list-matrix cell, at any depth in a list, unless jsonlite::unbox() wraps it. A data frame in a cell is sent as an array of objects whose values are not boxed. So a length-one atomic cell is sent as a one-element array, such as ["a"], and a NULL cell is sent as null. For example, a row with role "user", content "hi", and the cells "a", NULL, and list(k = "v") in a list-matrix column tags is sent as {"role":"user","content":"hi","tags":[["a"],null,{"k":["v"]}]}.
- No message, and no list or data frame inside a message or inside a cell or column of a data frame, has a name that is NA, empty, or repeated. jsonlite would send an NA or empty name as a number and rename a repeated name a to a.1. A list with no names passes. The names of a list column are not checked, because they are not sent. The names of an atomic vector and the column names of a matrix are not checked, because jsonlite sends those values as arrays.

The error for each rule above opens with "messages must be a data frame or an unnamed list of messages." and shows the form of a message. Four more rules are checked after them. Their error opens with "messages holds a field value that cannot be sent as JSON." and shows no form:

- No function sits inside messages: not as a field, in a list below a field, as a column of a data frame, or in a cell of a list column. jsonlite would send a function as its source text. A function with a class gets the error of this rule, not the error of a later one.
- Each value inside messages, in the places the rule above reads, is an atomic vector, a list, or NULL. An environment, a symbol, a call, a formula, an expression vector, an external pointer, or an S4 object breaks this rule, and

the error names its type. The rule reads the storage type, so a class set by hand on such a value does not hide it. jsonlite would send such a value as printed text, as `null`, or as an object or array of other data, or it would fail. A `NULL` field passes and is sent as `null`. An S4 object whose class contains an atomic type, such as `"numeric"`, passes this rule, and jsonlite sends its data part without its other slots. An S4 object whose class contains `"list"` passes, and the rule reads its elements. The rule does not read the class of a vector or a list. A vector with a class set by hand, such as `structure(1L, class = "NULL")`, passes, and jsonlite writes it by that class.

- No number inside `messages` is `NA`, `NaN`, `Inf`, or `-Inf`. The rule reads a double or integer vector with no class or with the class `"AsIs"` alone, such as a field wrapped in `I()`. jsonlite would send such a number as the string `"NA"`, `"NaN"`, `"Inf"`, or `"-Inf"`. The rule reads a field, an element of a field, a list at any depth below a field, a cell of a list column or a list-matrix column, and a matrix or array column. An atomic column with no `dim` attribute, of a data frame at any depth, is the exception. jsonlite leaves an `NA`, `NaN`, `Inf`, or `-Inf` cell of such a column out of its message. There the rule refuses `Inf` and `-Inf` alone, and an `NA` or `NaN` cell is left out as a missing field. A number with another class, such as a `Date`, is written by its class, so `as.Date(NA)` is sent as `null` and `as.Date(Inf)` as `"Inf"`. Below a message, the rule reads the parts of a list whose class is neither `"AsIs"` nor a data frame only when jsonlite writes that list as it writes the list without its class, such as a list with the class `c("foo", "list")`. It does not read the parts of a `POSIXlt` value.
- jsonlite can write the value, with the options that the request uses. If it cannot, the error gives the jsonlite message. For a data frame, jsonlite then writes each top-level column alone, in column order, as a data frame with the same row count. If a column fails, the error names the first that fails. For a column `d`, the line reads `Column "d" is the first column that jsonlite cannot write on its`. A list that is not a data frame gets the jsonlite message alone. This rule is checked last.

A list that is not a data frame is sent without its class attribute, and each of its messages is sent without its class attribute. A class on a field inside a message is kept, so a field wrapped in `I()` is sent as an array. A data frame and its columns keep their classes. A kept class that jsonlite has no method for, such as a field with the class `"foo"` alone, breaks the last rule.

The package checks the shape of `messages`. Of the field values, it refuses a function, a value that is not an atomic vector, a list, or `NULL`, a number with no class or with the class `"AsIs"` alone that jsonlite would send as a string or, in a data-frame column, would leave out because it is infinite, and a value that jsonlite cannot write. It does not check any other value of a role, the content, or another field of a message. The server checks them.

`host` Character. Server URL.

`logprobs` `TRUE` or `FALSE`. Whether to request logprobs. With `simplify = TRUE`, the `logprobs` of the result is `NULL`, because the package reads none from this endpoint. Any other value, `NULL` and `NA` included, aborts before the check for a running server.

`simplify` `TRUE` or `FALSE`. If `TRUE`, parses output to text. Any other value, `NULL` and `NA`

included, aborts before the check for a running server.

`...` Additional API arguments. A `response_format` here cannot be combined with `schema`. This function has no `store` argument, so a `store` here goes into the request body unchecked, whatever its value. The package checks a `stream` here. A `stream` other than `FALSE` or `NULL` aborts before the call checks for a running server, because the package reads a whole reply and not a streamed one.

`schema` A JSON Schema, written as a named list, that the reply must match, or `NULL` for a free text reply. It is sent as the `schema` field of a `response_format` of type `"json_schema"`, with the name `"response"` and `strict` set to `true`. A JSON array of one item must be written as a list, such as `required = list("score")`, or wrapped in `I()`. A plain vector of length one is sent as a single value, not as an array. An empty object nested in the schema, such as `properties`, is written `setNames(list(), character())`, because `list()` is sent as the empty array `[]`. The package checks only that `schema` is a named list, an empty list, or `NULL`. The server checks the schema itself.

`ttl` A whole number of seconds from 1 to `.Machine$integer.max`, or `NULL` to leave it out. It is how long the model stays loaded with no request. It has an effect only on a model that this request loads. The server loads a model that is not loaded yet when its just-in-time loading setting is on. A model that is already loaded keeps its idle time.

`token` Character or `NULL`. An API token for a server that requires authentication. `NULL` reads the `rlmstudio.token` option and then the `RLMSTUDIO_API_TOKEN` environment variable. See rlmstudio_token.

Value

If `simplify = FALSE`, returns a list representing the raw JSON response. A status-200 body that does not parse as JSON raises `rlmstudio_bad_response` with either setting of `simplify`. Otherwise, returns a character string containing the generated text. If `logprobs = TRUE`, it returns an `lms_chat_result` object whose `logprobs` is `NULL`, because the package reads no log probabilities from this endpoint. Such a call gives a warning of class `rlmstudio_no_logprobs` after the cut-off warning, if there is one. The `rlmstudio.quiet` option does not hide it. A call whose reply fails gives no such warning. Use `lms_chat_openresponses()` for log probabilities. With `simplify = TRUE`, reply content that is not one string, such as the `null` content of a reply that holds only a tool call, raises `rlmstudio_bad_response`.

With a `schema`, `simplify = TRUE`, and `logprobs = FALSE`, the reply is parsed with `jsonlite::parse_json(simplifyVecto = TRUE)` and the parsed value is returned. A JSON object becomes a named list, and an array of numbers becomes a vector. With `simplify = FALSE` or `logprobs = TRUE`, the reply stays a string. With a `schema`, `simplify = TRUE`, and `logprobs = FALSE`, a reply that a length limit ended raises `rlmstudio_bad_response`, also when it parses. With `simplify = TRUE` in any other setting, such a reply whose content is one string is returned with a warning. See the "Cut-off reply" section.

With `simplify = TRUE`, the reply is read from the first element of the `choices` field. No other element of `choices` is read. A request with `n` in `...` therefore returns the first choice alone, and the cut-off warning and abort depend on the finish reason of that choice alone. With `simplify = FALSE`, the body holds every choice.

With either setting of `simplify`, a reply from a model other than the one asked for raises `rlmstudio_model_mismatch`. See the "Reply from another model" section.

Server not running

Functions that call the LM Studio REST API open a TCP connection to the hostname and port named in `host` before they send the request. A function that checks its own arguments does that first, so a bad `model`, `job_id`, `input`, `inputs`, `messages`, `schema`, `ttl`, `previous_response_id`, or `batch_size`, a bad type of `list_models()` or `list_instances()`, a bad `TRUE` or `FALSE` argument such as `simplify`, `logprobs`, `quiet`, `force`, or `store`, a `store` on the `"openai"` route of `lms_chat()` or `lms_chat_batch()`, a `stream` in the `...` of a chat function, a `logprobs` in the `...` of `lms_chat_batch()` or `lms_chat_native()`, a value in the `...` of `lms_chat_batch()` that `lms_chat()` reads as an argument already given, an `instructions` or `messages` in the `...` of `lms_chat()` or `lms_chat_batch()` on the route where `lms_chat()` sets it, or a name, id, or type string that is not valid in its encoding or is marked as bytes, aborts with an argument message and no condition class even when the server is down. A condition of class `rlmstudio_no_server` is raised when that connection cannot be opened. A refused connection raises it. So do an address the package cannot parse and a hostname that does not resolve. An address that neither accepts nor refuses the connection also raises it. That case waits for the operating system to give up, which can take a minute. If the hostname of `host` is `localhost`, in any letter case, or `127.0.0.1`, start the server with `lms_server_start()` and the port of `host`. For the default `host`, that is `lms_server_start(port = 1234)`. With no `port`, LM Studio uses the port that it used last. For any other `host`, start the server on the machine that `host` names, or give `host` the address that your server listens on. A `host` that does not parse gets this second hint too. The condition message gives the same hint for the `host` of the call.

The check reads the port and nothing else. Any process holding that port accepts the connection, so the condition is not raised even though no LM Studio server is there. The call then does not raise `rlmstudio_no_server`, and what it does depends on what answers. On a status-200 body that does not parse as JSON, the chat functions, `lms_embed()`, `list_models()`, `list_instances()`, `lms_load()`, `lms_download()`, `lms_download_status()`, and `lms_unload_all()` raise `rlmstudio_bad_response`. `lms_unload()` does not read the body, so it can report success. A body that parses as JSON but has another shape can come back unchanged with `simplify = FALSE`. With `simplify = TRUE`, the chat functions and `lms_embed()` raise `rlmstudio_bad_response` for it. `list_models()` and `list_instances()` raise it for a model list with another shape, and so do `lms_unload_all()` and `lms_load()` without `force = TRUE`, which read that list. `lms_chat_openai()` and `lms_chat_openresponses()` raise it for such a model list too, with either setting of `simplify`, through the model lookup that a reply from another model starts. `lms_chat()` raises it through them. `lms_embed()` raises it in the same way for the inputs of that request. `lms_load()`, `lms_download()`, and `lms_download_status()` raise it for a reply of their own with another shape, such as `{}`. A process that does not answer in HTTP gives an `httr2_failure` error. Use `lms_server_ready()` for the stronger test: it asks the host for a model list and reports `TRUE` only for a model list that `list_models()` can read.

`lms_chat_batch()` checks the server once before its first input, and `lms_chat()` checks it again for each input. If that check finds the server gone during the batch, the batch aborts with `rlmstudio_no_server`, and no request goes out after that. The condition then carries a `results` field, a list as long as `inputs`. Its elements before the lost input hold the values that `format = "list"` returns for those inputs. The element of the lost input and every element after it are `NULL`. The check before the first input adds no `results` field. A connection that fails after the check passes, such as a server that stops during a request, raises an `httr2_failure` error instead. That error aborts the batch and carries no `results` field.

`lms_embed()` checks the server before each request, and a request carries at most `batch_size` inputs. If a check after the first request, or the check of a model-list lookup, finds the server gone,

the call aborts with rlmstudio_no_server. Once a request has succeeded, the condition carries a results field. With simplify = TRUE, results is a matrix with NA in each row whose embedding did not arrive. With simplify = FALSE, it is a list with one element per batch, with NULL in the element of the request that ended the call and in every element after it.

API failure

A condition of class rlmstudio_api_error is raised when a REST call returns a response that the wrapper treats as a failure. The condition carries a status field, which holds the HTTP response status as an integer. It also carries a code field. The field holds the string at error.code of the response body, such as "model_not_found". It is NULL when the body does not parse, when error is not a JSON object, or when its code is not one string. A server_message field holds the error text as the server sent it: the error string of the body, or else the string at error.message. It is NULL when the body does not parse or holds neither string.

On LM Studio 0.4.26+4 with no model loaded, /v1/chat/completions, /v1/responses, and /v1/embeddings answered a model name that the server could not find with status 400, no code, and a server_message that starts with "No models loaded".

lms_chat_batch() aborts on it when its status is 401, 403, or 404, when its status is 400 and its code is "model_not_found", and when its status is 400 and its server_message starts with "No models loaded", in that letter case. No request goes out after that input. The condition then carries a results field that follows the rule for a lost server in the "Server not running" section: its elements before the failed input hold the values that format = "list" returns for those inputs, and the element of the failed input and every element after it are NULL. For any other status, the element of the failed input holds the condition, or NA where the result is text, and the batch warns once and goes on. See the details of lms_chat_batch().

lms_embed() follows the same rule for each request, except for the code "model_not_found". A 401, 403, or 404 aborts the call, and so does a 400 whose server_message starts with "No models loaded". Once a request has succeeded, the condition carries a results field, a matrix or a list as the "Server not running" section describes. Any other status, a 400 with the code "model_not_found" included, fails the inputs of that request alone, and the call warns once and goes on. If every request fails, the call aborts with the first condition. See the details of lms_embed().

Malformed response

A condition of class rlmstudio_bad_response is raised when the server answers with a status the wrapper accepts and a body the wrapper cannot read. It is raised where a wrapper checks the body before it reshapes it, rather than indexing straight into whatever arrived. Eleven functions raise it for a body they cannot read: lms_embed(), lms_chat(), lms_chat_native(), lms_chat_openresponses(), lms_chat_openai(), list_models(), list_instances(), lms_load(), lms_download(), lms_download_status(), and lms_unload_all(). lms_chat() raises it through the chat function it calls. lms_unload_all() raises it through list_models(), and so does lms_load() unless force = TRUE. lms_chat_batch() raises it only as rlmstudio_model_mismatch, which the "Reply from another model" section of lms_chat_openai() describes.

All eleven raise it for a status-200 body that does not parse as JSON, such as an HTML page from a proxy, JSON text that stops part way, or an empty body. In the functions that take simplify, the body is parsed before simplify is read, so the condition is raised whatever simplify is. The body is parsed by its content and not by its Content-Type header, so valid JSON under text/plain is read as JSON. The body is read as JSON text and nothing else. A body whose text is a URL

or the path of a file does not parse, and the package does not fetch the URL or read the file. The message says that the body did not parse as JSON and that something other than LM Studio may be answering on the host. It does not hold the body text.

The condition carries a `status` field, which holds the HTTP response status as an integer. Today the status is always 200: each of these functions reads the body only after a 200, and reports every other status as an `rlmstudio_api_error` instead.

Malformed model list

`list_models()` also raises it for a status-200 model list with the wrong shape. `lms_unload_all()` and `lms_load()` without `force = TRUE` raise it through `list_models()`. `lms_chat_openai()`, `lms_chat_openresponses()`, and `lms_embed()` apply these rules to the model list of their model lookup, as the "Reply from another model" section of `lms_chat_openai()` describes. A model list must follow four rules. Each field is read by its exact name, so a field named `keyX` does not stand in for `key`.

1. The body is a JSON object whose `models` field is an array. The array can be empty.
2. Each entry of `models` is a JSON object. Its `type` and `key` are strings, and its `loaded_instances` is an array.
3. The `size_bytes` of an entry is a number, or absent, or `null`. So is its `max_context_length`.
4. Each entry of `loaded_instances` is a JSON object whose `id` is a string with a character that is not whitespace.

The rules are checked before the `type` and `loaded` filters, so an entry that the filters drop can still raise the condition. The message names the field or entry that broke a rule. `lms_server_ready()` applies the same rules and returns `FALSE` for a body that breaks one.

`list_instances()` applies the four rules too, and three more. They cover only an entry whose `type` is in its `type` argument and that has at least one loaded instance. The `display_name` of such an entry is a string, or absent, or `null`. The `remaining_ttl_seconds` of each of its instances is a number, or absent, or `null`. The `config` of each of its instances is a JSON object, or absent, or `null`. `list_models()` and `lms_server_ready()` do not apply these three rules.

Malformed chat reply

`lms_chat_native()` and `lms_chat_openresponses()` raise it with `simplify = TRUE` when the reply holds no readable answer text. Both read the answer from the items of type `"message"` in the `output` array. They raise it when `output` is missing, empty, or not an array, or when an item in it is not a JSON object. They also raise it when no item has the type `"message"`, as in a reply that holds only reasoning or a tool call. The text of a message must be one string. For `lms_chat_native()` that is the `content` of the item. For `lms_chat_openresponses()` it is the `text` of each part of type `"output_text"`. For `lms_chat_openresponses()` only, the `content` of each message must be an array of JSON objects, and the messages together must hold at least one `"output_text"` part.

Apart from a body that does not parse as JSON, `lms_chat_openai()` raises it in four cases, all only with `simplify = TRUE`. The first case is a response whose `choices` field is missing, empty, or not an array, or whose first element is not a JSON object with a `message` object in it, so there is no reply to read. This case is raised with or without a `schema`, and with `logprobs = TRUE` as well. The second case is a reply that does not parse. A `schema` was given, `logprobs = FALSE`, and the reply content is not one string of valid JSON. The third case is reply content that is not one string, such as

null, a missing content field, a number, or an array. A reply that holds only a tool call has null content. This case is raised without a schema, and with logprobs = TRUE with or without one. The fourth case is a cut-off reply. A schema was given, logprobs = FALSE, and the server reports the finish reason "length". This case is raised also when the reply content parses, because a reply that stops part way can parse to a wrong value, such as the first digit of a longer number. In the second, third, and fourth cases, if the server reports the finish reason "length", a length limit ended the reply. The limit is max_tokens or the context length of the model. The message then says so and names both.

With simplify = TRUE, lms_chat_native(), lms_chat_openresponses(), and lms_chat_openai() also raise it for a body that is a bare JSON value, such as 5, "s", or true. The message says that the response body is not a JSON object. A body of null gets the message about its missing output or choices field instead. lms_chat() can raise the condition through all three chat functions.

lms_chat_batch() does not abort on it, except on the subclass rlmstudio_model_mismatch, as the "Reply from another model" section of lms_chat_openai() says. The element of the failed input holds the condition, or NA where the result is text, and the batch warns once and goes on. See the details of lms_chat_batch().

A condition from lms_chat_openai() about its reply also carries two more fields. A condition from the model-list lookup does not, as the "Reply from another model" section of lms_chat_openai() says. The content field holds the reply content of the first choice, and the finish_reason field holds the finish reason of the first choice. Both are NULL for a response with no choices. In the third case, content holds the value that was read, which is NULL for null or missing content. For the second, third, and fourth cases, the message names the content field, so you can read what the model wrote without a second request. The other messages of the chat functions about the reply name simplify = FALSE, which returns the body unchanged, with one exception. A body that did not parse as JSON is checked before that argument is read, so its message points at the host instead. For such a body, the content and finish_reason fields of a condition from lms_chat_openai() are NULL. The messages of rlmstudio_model_mismatch and of the model-list lookup do not name simplify = FALSE, because the check runs with either setting of simplify.

Cut-off reply

A warning of class rlmstudio_reply_cut_off is given when a length limit ended a reply that lms_chat_openai() returns as text. The server reports this with the finish reason "length". The limit is max_tokens or the context length of the model, and the reply does not say which one. The message names both. The warning is given with simplify = TRUE for reply content that is one string, in three settings: no schema, logprobs = TRUE, and a schema with logprobs = TRUE. The call returns what the same reply returns without the cut-off. A schema reply with logprobs = FALSE raises rlmstudio_bad_response instead, as the "Malformed chat reply" section says. The warning and that abort read the finish reason of the first choice, which is the choice that the call reads. No other element of choices is read. With simplify = FALSE, the call returns the body with every choice and no warning.

lms_chat() gives the warning through lms_chat_openai() with api_type = "openai". lms_chat_batch() gives one warning of this class for the whole batch in place of one for each input. It names the count and the positions of the cut-off inputs, and each of those elements keeps its reply. It comes after the warning about failed inputs. A batch that aborts with a results field gives this warning before the abort. It names the cut-off inputs whose replies the results field of the abort holds. The native and OpenResponses routes give no such warning.

The warning shows whatever quiet and the rlmstudio.quiet option say, because it is the only sign that an answer is not complete.

Reply from another model

lms_chat_openai(), lms_chat_openresponses(), and lms_embed() read the model field of each status-200 reply, with either setting of simplify. On LM Studio 0.4.25+1 with one chat model loaded, both chat routes answered a model name that the server could not find with status 200 and a reply from the loaded model. With two chat models loaded, they answered with status 400 and the code "model_not_found", as the "API failure" section describes. On LM Studio 0.4.26+4 with one embedding model loaded, /v1/embeddings answered a name that it could not find with status 200 and vectors from the loaded embedding model. The model field names the loaded instance that answered. Its id can differ from the key that the call asked for.

If the model field equals the name that the call sent, the reply is accepted. If it differs, the call sends one request for the model list to api/v1/models on the same host, with the token of the call, and prints no message. The reply is accepted if a model whose key equals the asked name in any letter case has a loaded instance whose id is the model field. Otherwise the call aborts with a condition of class rlmstudio_model_mismatch. That condition is also an rlmstudio_bad_response, and its status is 200. Its model field holds the asked name, and its reply_model field holds the model field of the reply. The message names both. A condition from lms_chat_openai() also carries the content and finish_reason fields, both NULL.

Within one call of lms_chat_batch() or lms_embed(), a reply whose model field a lookup of that call already accepted for the asked name and the same host is accepted with no other request. Each such call, and each call of lms_chat_openai() or lms_chat_openresponses() outside a batch, starts with no accepted ids.

The check runs before the reply is read, so it also runs with logprobs = TRUE, with a schema on lms_chat_openai(), and for a reply with no answer text. In lms_embed(), it runs before the vectors are read. A reply is not checked if its body is not a JSON object, if it has no model field, or if its model is not one string or holds only whitespace. A body that is not a JSON object is returned with simplify = FALSE, and with simplify = TRUE it raises the error that the "Malformed chat reply" section describes, or for lms_embed() the "Malformed embeddings" section. If the instance that answered is unloaded before the model-list request, the call aborts, also when that instance belongs to the asked model.

If the model-list request fails, the check raises the condition of that failure. The message opens with the label of the function, followed by "because the model-list lookup failed". A status other than 200 raises rlmstudio_api_error. A body that does not parse as JSON, or that breaks a rule of a model list in the "Malformed model list" section, raises rlmstudio_bad_response. A server that the port check before the request cannot reach raises rlmstudio_no_server. A condition from the lookup does not carry the content and finish_reason fields, also when lms_chat_openai() raises it.

lms_chat() runs the check through lms_chat_openai() and lms_chat_openresponses(). lms_chat_native() does not check the reply. On LM Studio 0.4.25+1, /api/v1/chat answered a name that it could not find with status 404, which raises rlmstudio_api_error.

lms_chat_batch() aborts at the first input that raises rlmstudio_model_mismatch, on the "openai" and "openresponses" routes. It also aborts at an rlmstudio_api_error with status 400 whose code is "model_not_found", and at one with status 400 whose server_message starts with "No

models loaded". No request goes out after that input. In each case, the condition carries a `results` field that follows the rule for a lost server in the "Server not running" section.

`lms_embed()` aborts at the first request that raises `rlmstudio_model_mismatch`, and no request goes out after it. A failed model-list lookup that raises `rlmstudio_no_server`, or an `rlmstudio_api_error` that would end the call as an embeddings failure, such as one with status 401, 403, or 404, ends the call in the same way. Once a request has succeeded, the condition carries a `results` field, a matrix or a list as the "Server not running" section describes. Any other `rlmstudio_api_error` or `rlmstudio_bad_response` from the lookup fails the inputs of that request alone, as the "API failure" section describes for a failed request. An error of any other class, such as an `httr2_failure` from a server that stops during the lookup, aborts the call unchanged.

Examples

```
## Not run:
lms_chat_openai(
  model = "google/gemma-3-1b",
  messages = list(
    list(role = "user", content = "Rate 'Great value.' from 1 to 5.")
  ),
  schema = list(
    type = "object",
    properties = list(score = list(type = "integer")),
    required = list("score")
  )
)

## End(Not run)
```

---

lms_chat_openresponses

Chat Completion via OpenResponses API

Description

Direct interface to LM Studio's OpenResponses endpoint. Supports logprobs and custom instructions.

Usage

```
lms_chat_openresponses(
  model,
  input,
  instructions = NULL,
  host = "http://localhost:1234",
  logprobs = FALSE,
  simplify = TRUE,
  ...,
  previous_response_id = NULL,
```

```
  store = NULL,
  token = NULL
)
```

Arguments

model	Character. The loaded model name. Must be one name, given as a single string. The string must be valid in its declared encoding and not marked "bytes". A class, names, and the S4 bit are removed before the name is sent.
input	Character. The user prompt, as one string. A character vector must hold no missing values, and its length must be one. Any other character value aborts before the check for a running server. To send several prompts, one request each, use lms_chat_batch(). A list is sent as given in the input field.
instructions	Character. Optional system instructions.
host	Character. Server URL.
logprobs	TRUE or FALSE. Whether to return token probabilities. Any other value, NULL and NA included, aborts before the check for a running server.
simplify	TRUE or FALSE. If TRUE, parses output to text and dataframe. If FALSE, returns raw list. Any other value, NULL and NA included, aborts before the check for a running server.
...	Additional API arguments (e.g., top_logprobs, temperature). This endpoint accepts a ttl field and ignores it. The model keeps the idle time that the server sets. The package checks a stream here. A stream other than FALSE or NULL aborts before the call checks for a running server, because the package reads a whole reply and not a streamed one.
previous_response_id	One string, a value that carries a response_id attribute, or NULL. It names the stored reply that this chat continues. Pass an earlier reply of this function or of lms_chat_native() itself, such as first, or its id, attr(first, "response_id"). A value that carries a response_id attribute sends that attribute in place of the value, also when the value is a list or is itself an id. Only that exact attribute name is read. A value with no such attribute is sent as the id itself. So a reply that came back with no id, such as a native reply sent with store = FALSE, goes out as its own text, and the call raises rlmstudio_api_error with status 400. A reply text that breaks the id rules below, such as an empty text or a text of whitespace only, aborts before the request, as such an id does. The character vector that lms_chat_batch() returns with format = "vector" and the output column of its data frame carry no attribute. The data frame holds the ids in its response_id column. The id, the attribute when there is one, must be valid in its declared encoding and not marked "bytes". A class, names, and the S4 bit are removed before the id is sent. NULL, the default, starts a new thread. As the id, NA, an empty string, a string of whitespace only, a value that is not a string, and more or fewer than one string abort before the check for a running server. For an attribute, the message names the attribute. Only this exact argument name is checked. A shortened name, such as previous, goes into the request body unchecked, under the name you wrote. An id that the server does not hold raises rlmstudio_api_error

with status 400 and the code "previous_response_not_found". lms_chat() and lms_chat_batch() refuse an id or a value that carries one with api_type = "openai", because the OpenAI chat endpoint keeps no thread. lms_chat_openai() has no such argument. A previous_response_id in its ... goes into the request body unchecked, and the endpoint ignores it.

store — TRUE, FALSE, or NULL. Whether the server stores the reply, so that a later call can continue from its id. NULL, the default, sends no store field, so the server default applies, and that default stores the reply. TRUE and FALSE go out as a plain JSON true or false, also when the value has names, dimensions, or a class. Any other value, NA included, aborts before the check for a running server. Only this exact name is checked. A shortened name, such as sto, goes into the request body unchecked, under the name you wrote. lms_chat() and lms_chat_batch() refuse TRUE or FALSE with api_type = "openai".

token — Character or NULL. An API token for a server that requires authentication. NULL reads the rlmstudio.token option and then the RLMSTUDIO_API_TOKEN environment variable. See rlmstudio_token.

Value

If simplify = FALSE, returns a list representing the raw JSON response. A status-200 body that does not parse as JSON raises rlmstudio_bad_response with either setting of simplify. Otherwise, returns one character string: the text of every part of type "output_text" in the items of type "message", pasted together in order with no separator. Reasoning items, tool calls, and parts of other types, such as a refusal, are skipped. If logprobs = TRUE and at least one of those parts carries log probabilities, returns an object of class lms_chat_result with that text and a data frame of the probabilities of every such part, in order. If no part carries them, it returns the string. A reply with no readable answer text raises rlmstudio_bad_response, as described below.

The data frame has one row for each candidate in the top_logprobs of each step, or one row with NA candidates for a step with none. It has five columns. step_token and step_logprob hold the token and the log probability of the step. candidate_token and candidate_logprob hold those of the candidate. step is an integer that numbers the steps from 1 in reply order, across all the parts. Each row of a step has the number of that step, so two steps with the same token keep different numbers. Each field is read by its exact name. A field that is null or absent gives NA, and so does a field whose name only starts with the one asked for, such as tokenX. With logprobs = TRUE, the logprobs value of each "output_text" part must follow seven rules, which the "Malformed logprobs" section below lists. A value that breaks one raises rlmstudio_bad_response, and the message names the first broken rule in the order the section below gives. Parts of other types are not checked. With logprobs = FALSE, the value is not read, and the call returns the text.

With simplify = TRUE, the string or the lms_chat_result carries the id field of the reply in a response_id attribute. Pass the value itself as previous_response_id to continue the thread, and the attribute is sent. The value has no attribute when id is absent or is not one string. A reply sent with store = FALSE still has an id, so its value carries the attribute, but the server does not hold that reply. A later call of this function that passes that id as previous_response_id raises rlmstudio_api_error with status 400 and the code "previous_response_not_found". lms_chat_native() reads the attribute from the response_id field of its reply instead. A native reply sent with store = FALSE has no response_id field, so its value has no attribute.

With either setting of simplify, a reply from a model other than the one asked for raises rlmstudio_model_mismatch. See the "Reply from another model" section.

Server not running

Functions that call the LM Studio REST API open a TCP connection to the hostname and port named in `host` before they send the request. A function that checks its own arguments does that first, so a bad `model`, `job_id`, `input`, `inputs`, `messages`, `schema`, `ttl`, `previous_response_id`, or `batch_size`, a bad type of `list_models()` or `list_instances()`, a bad `TRUE` or `FALSE` argument such as `simplify`, `logprobs`, `quiet`, `force`, or `store`, a `store` on the `"openai"` route of `lms_chat()` or `lms_chat_batch()`, a `stream` in the `...` of a chat function, a `logprobs` in the `...` of `lms_chat_batch()` or `lms_chat_native()`, a value in the `...` of `lms_chat_batch()` that `lms_chat()` reads as an argument already given, an `instructions` or `messages` in the `...` of `lms_chat()` or `lms_chat_batch()` on the route where `lms_chat()` sets it, or a name, id, or type string that is not valid in its encoding or is marked as bytes, aborts with an argument message and no condition class even when the server is down. A condition of class `rlmstudio_no_server` is raised when that connection cannot be opened. A refused connection raises it. So do an address the package cannot parse and a hostname that does not resolve. An address that neither accepts nor refuses the connection also raises it. That case waits for the operating system to give up, which can take a minute. If the hostname of `host` is `localhost`, in any letter case, or `127.0.0.1`, start the server with `lms_server_start()` and the port of `host`. For the default `host`, that is `lms_server_start(port = 1234)`. With no `port`, LM Studio uses the port that it used last. For any other `host`, start the server on the machine that `host` names, or give `host` the address that your server listens on. A `host` that does not parse gets this second hint too. The condition message gives the same hint for the `host` of the call.

The check reads the port and nothing else. Any process holding that port accepts the connection, so the condition is not raised even though no LM Studio server is there. The call then does not raise `rlmstudio_no_server`, and what it does depends on what answers. On a status-200 body that does not parse as JSON, the chat functions, `lms_embed()`, `list_models()`, `list_instances()`, `lms_load()`, `lms_download()`, `lms_download_status()`, and `lms_unload_all()` raise `rlmstudio_bad_response`. `lms_unload()` does not read the body, so it can report success. A body that parses as JSON but has another shape can come back unchanged with `simplify = FALSE`. With `simplify = TRUE`, the chat functions and `lms_embed()` raise `rlmstudio_bad_response` for it. `list_models()` and `list_instances()` raise it for a model list with another shape, and so do `lms_unload_all()` and `lms_load()` without `force = TRUE`, which read that list. `lms_chat_openai()` and `lms_chat_openresponses()` raise it for such a model list too, with either setting of `simplify`, through the model lookup that a reply from another model starts. `lms_chat()` raises it through them. `lms_embed()` raises it in the same way for the inputs of that request. `lms_load()`, `lms_download()`, and `lms_download_status()` raise it for a reply of their own with another shape, such as `{}`. A process that does not answer in HTTP gives an `httr2_failure` error. Use `lms_server_ready()` for the stronger test: it asks the host for a model list and reports `TRUE` only for a model list that `list_models()` can read.

`lms_chat_batch()` checks the server once before its first input, and `lms_chat()` checks it again for each input. If that check finds the server gone during the batch, the batch aborts with `rlmstudio_no_server`, and no request goes out after that. The condition then carries a `results` field, a list as long as `inputs`. Its elements before the lost input hold the values that `format = "list"` returns for those inputs. The element of the lost input and every element after it are `NULL`. The check before the first input adds no `results` field. A connection that fails after the check passes, such as a server that stops during a request, raises an `httr2_failure` error instead. That error aborts the batch and carries no `results` field.

`lms_embed()` checks the server before each request, and a request carries at most `batch_size` inputs. If a check after the first request, or the check of a model-list lookup, finds the server gone,

the call aborts with rlmstudio_no_server. Once a request has succeeded, the condition carries a results field. With simplify = TRUE, results is a matrix with NA in each row whose embedding did not arrive. With simplify = FALSE, it is a list with one element per batch, with NULL in the element of the request that ended the call and in every element after it.

API failure

A condition of class rlmstudio_api_error is raised when a REST call returns a response that the wrapper treats as a failure. The condition carries a status field, which holds the HTTP response status as an integer. It also carries a code field. The field holds the string at error.code of the response body, such as "model_not_found". It is NULL when the body does not parse, when error is not a JSON object, or when its code is not one string. A server_message field holds the error text as the server sent it: the error string of the body, or else the string at error.message. It is NULL when the body does not parse or holds neither string.

On LM Studio 0.4.26+4 with no model loaded, /v1/chat/completions, /v1/responses, and /v1/embeddings answered a model name that the server could not find with status 400, no code, and a server_message that starts with "No models loaded".

lms_chat_batch() aborts on it when its status is 401, 403, or 404, when its status is 400 and its code is "model_not_found", and when its status is 400 and its server_message starts with "No models loaded", in that letter case. No request goes out after that input. The condition then carries a results field that follows the rule for a lost server in the "Server not running" section: its elements before the failed input hold the values that format = "list" returns for those inputs, and the element of the failed input and every element after it are NULL. For any other status, the element of the failed input holds the condition, or NA where the result is text, and the batch warns once and goes on. See the details of lms_chat_batch().

lms_embed() follows the same rule for each request, except for the code "model_not_found". A 401, 403, or 404 aborts the call, and so does a 400 whose server_message starts with "No models loaded". Once a request has succeeded, the condition carries a results field, a matrix or a list as the "Server not running" section describes. Any other status, a 400 with the code "model_not_found" included, fails the inputs of that request alone, and the call warns once and goes on. If every request fails, the call aborts with the first condition. See the details of lms_embed().

Malformed response

A condition of class rlmstudio_bad_response is raised when the server answers with a status the wrapper accepts and a body the wrapper cannot read. It is raised where a wrapper checks the body before it reshapes it, rather than indexing straight into whatever arrived. Eleven functions raise it for a body they cannot read: lms_embed(), lms_chat(), lms_chat_native(), lms_chat_openresponses(), lms_chat_openai(), list_models(), list_instances(), lms_load(), lms_download(), lms_download_status(), and lms_unload_all(). lms_chat() raises it through the chat function it calls. lms_unload_all() raises it through list_models(), and so does lms_load() unless force = TRUE. lms_chat_batch() raises it only as rlmstudio_model_mismatch, which the "Reply from another model" section of lms_chat_openai() describes.

All eleven raise it for a status-200 body that does not parse as JSON, such as an HTML page from a proxy, JSON text that stops part way, or an empty body. In the functions that take simplify, the body is parsed before simplify is read, so the condition is raised whatever simplify is. The body is parsed by its content and not by its Content-Type header, so valid JSON under text/plain is read as JSON. The body is read as JSON text and nothing else. A body whose text is a URL

or the path of a file does not parse, and the package does not fetch the URL or read the file. The message says that the body did not parse as JSON and that something other than LM Studio may be answering on the host. It does not hold the body text.

The condition carries a `status` field, which holds the HTTP response status as an integer. Today the status is always 200: each of these functions reads the body only after a 200, and reports every other status as an `rlmstudio_api_error` instead.

Malformed model list

`list_models()` also raises it for a status-200 model list with the wrong shape. `lms_unload_all()` and `lms_load()` without `force = TRUE` raise it through `list_models()`. `lms_chat_openai()`, `lms_chat_openresponses()`, and `lms_embed()` apply these rules to the model list of their model lookup, as the "Reply from another model" section of `lms_chat_openai()` describes. A model list must follow four rules. Each field is read by its exact name, so a field named `keyX` does not stand in for `key`.

1. The body is a JSON object whose `models` field is an array. The array can be empty.
2. Each entry of `models` is a JSON object. Its `type` and `key` are strings, and its `loaded_instances` is an array.
3. The `size_bytes` of an entry is a number, or absent, or `null`. So is its `max_context_length`.
4. Each entry of `loaded_instances` is a JSON object whose `id` is a string with a character that is not whitespace.

The rules are checked before the `type` and `loaded` filters, so an entry that the filters drop can still raise the condition. The message names the field or entry that broke a rule. `lms_server_ready()` applies the same rules and returns `FALSE` for a body that breaks one.

`list_instances()` applies the four rules too, and three more. They cover only an entry whose `type` is in its `type` argument and that has at least one loaded instance. The `display_name` of such an entry is a string, or absent, or `null`. The `remaining_ttl_seconds` of each of its instances is a number, or absent, or `null`. The `config` of each of its instances is a JSON object, or absent, or `null`. `list_models()` and `lms_server_ready()` do not apply these three rules.

Malformed chat reply

`lms_chat_native()` and `lms_chat_openresponses()` raise it with `simplify = TRUE` when the reply holds no readable answer text. Both read the answer from the items of type `"message"` in the `output` array. They raise it when `output` is missing, empty, or not an array, or when an item in it is not a JSON object. They also raise it when no item has the type `"message"`, as in a reply that holds only reasoning or a tool call. The text of a message must be one string. For `lms_chat_native()` that is the `content` of the item. For `lms_chat_openresponses()` it is the `text` of each part of type `"output_text"`. For `lms_chat_openresponses()` only, the `content` of each message must be an array of JSON objects, and the messages together must hold at least one `"output_text"` part.

Apart from a body that does not parse as JSON, `lms_chat_openai()` raises it in four cases, all only with `simplify = TRUE`. The first case is a response whose `choices` field is missing, empty, or not an array, or whose first element is not a JSON object with a `message` object in it, so there is no reply to read. This case is raised with or without a `schema`, and with `logprobs = TRUE` as well. The second case is a reply that does not parse. A `schema` was given, `logprobs = FALSE`, and the reply content is not one string of valid JSON. The third case is reply content that is not one string, such as

null, a missing content field, a number, or an array. A reply that holds only a tool call has null content. This case is raised without a schema, and with logprobs = TRUE with or without one. The fourth case is a cut-off reply. A schema was given, logprobs = FALSE, and the server reports the finish reason "length". This case is raised also when the reply content parses, because a reply that stops part way can parse to a wrong value, such as the first digit of a longer number. In the second, third, and fourth cases, if the server reports the finish reason "length", a length limit ended the reply. The limit is max_tokens or the context length of the model. The message then says so and names both.

With simplify = TRUE, lms_chat_native(), lms_chat_openresponses(), and lms_chat_openai() also raise it for a body that is a bare JSON value, such as 5, "s", or true. The message says that the response body is not a JSON object. A body of null gets the message about its missing output or choices field instead. lms_chat() can raise the condition through all three chat functions.

lms_chat_batch() does not abort on it, except on the subclass rlmstudio_model_mismatch, as the "Reply from another model" section of lms_chat_openai() says. The element of the failed input holds the condition, or NA where the result is text, and the batch warns once and goes on. See the details of lms_chat_batch().

A condition from lms_chat_openai() about its reply also carries two more fields. A condition from the model-list lookup does not, as the "Reply from another model" section of lms_chat_openai() says. The content field holds the reply content of the first choice, and the finish_reason field holds the finish reason of the first choice. Both are NULL for a response with no choices. In the third case, content holds the value that was read, which is NULL for null or missing content. For the second, third, and fourth cases, the message names the content field, so you can read what the model wrote without a second request. The other messages of the chat functions about the reply name simplify = FALSE, which returns the body unchanged, with one exception. A body that did not parse as JSON is checked before that argument is read, so its message points at the host instead. For such a body, the content and finish_reason fields of a condition from lms_chat_openai() are NULL. The messages of rlmstudio_model_mismatch and of the model-list lookup do not name simplify = FALSE, because the check runs with either setting of simplify.

Malformed logprobs

With simplify = TRUE and logprobs = TRUE, lms_chat_openresponses() also raises it for a logprobs value that breaks one of these rules. The logprobs value of each "output_text" part is checked. A logprobs value, token, logprob, or top_logprobs that is null or absent passes its rule. A null step or candidate breaks rule 2 or rule 6.

1. The value is an array.
2. Each step in the array is a JSON object.
3. The token of a step is a string.
4. The logprob of a step is a number.
5. The top_logprobs of a step is an array.
6. Each candidate in top_logprobs is a JSON object.
7. The token and logprob of each candidate follow rules 3 and 4.

The parts are checked in order, then the steps of a part, then the candidates of a step, one at a time. Within a step, rules 3, 4, and 5 come before the candidates. The message names the first broken rule that this order reaches. These checks run only after the text of every "output_text" part is

read, so a reply that also has a bad `text` in any part gets the text message. Parts of other types, such as a refusal, are not checked, and with `logprobs = FALSE` no part is checked. Fields are read by their exact names, so a field whose name only starts with the one asked for, such as `tokenX`, reads as absent and gives `NA` in the data frame.

Reply from another model

`lms_chat_openai()`, `lms_chat_openresponses()`, and `lms_embed()` read the `model` field of each status-200 reply, with either setting of `simplify`. On LM Studio 0.4.25+1 with one chat model loaded, both chat routes answered a model name that the server could not find with status 200 and a reply from the loaded model. With two chat models loaded, they answered with status 400 and the `code` `"model_not_found"`, as the "API failure" section describes. On LM Studio 0.4.26+4 with one embedding model loaded, `/v1/embeddings` answered a name that it could not find with status 200 and vectors from the loaded embedding model. The `model` field names the loaded instance that answered. Its id can differ from the key that the call asked for.

If the `model` field equals the name that the call sent, the reply is accepted. If it differs, the call sends one request for the model list to `api/v1/models` on the same host, with the token of the call, and prints no message. The reply is accepted if a model whose key equals the asked name in any letter case has a loaded instance whose id is the `model` field. Otherwise the call aborts with a condition of class `rlmstudio_model_mismatch`. That condition is also an `rlmstudio_bad_response`, and its `status` is 200. Its `model` field holds the asked name, and its `reply_model` field holds the `model` field of the reply. The message names both. A condition from `lms_chat_openai()` also carries the `content` and `finish_reason` fields, both `NULL`.

Within one call of `lms_chat_batch()` or `lms_embed()`, a reply whose `model` field a lookup of that call already accepted for the asked name and the same host is accepted with no other request. Each such call, and each call of `lms_chat_openai()` or `lms_chat_openresponses()` outside a batch, starts with no accepted ids.

The check runs before the reply is read, so it also runs with `logprobs = TRUE`, with a `schema` on `lms_chat_openai()`, and for a reply with no answer text. In `lms_embed()`, it runs before the vectors are read. A reply is not checked if its body is not a JSON object, if it has no `model` field, or if its `model` is not one string or holds only whitespace. A body that is not a JSON object is returned with `simplify = FALSE`, and with `simplify = TRUE` it raises the error that the "Malformed chat reply" section describes, or for `lms_embed()` the "Malformed embeddings" section. If the instance that answered is unloaded before the model-list request, the call aborts, also when that instance belongs to the asked model.

If the model-list request fails, the check raises the condition of that failure. The message opens with the label of the function, followed by "because the model-list lookup failed". A status other than 200 raises `rlmstudio_api_error`. A body that does not parse as JSON, or that breaks a rule of a model list in the "Malformed model list" section, raises `rlmstudio_bad_response`. A server that the port check before the request cannot reach raises `rlmstudio_no_server`. A condition from the lookup does not carry the `content` and `finish_reason` fields, also when `lms_chat_openai()` raises it.

`lms_chat()` runs the check through `lms_chat_openai()` and `lms_chat_openresponses()`. `lms_chat_native()` does not check the reply. On LM Studio 0.4.25+1, `/api/v1/chat` answered a name that it could not find with status 404, which raises `rlmstudio_api_error`.

`lms_chat_batch()` aborts at the first input that raises `rlmstudio_model_mismatch`, on the `"openai"` and `"openresponses"` routes. It also aborts at an `rlmstudio_api_error` with status 400 whose

code is "model_not_found", and at one with status 400 whose server_message starts with "No models loaded". No request goes out after that input. In each case, the condition carries a results field that follows the rule for a lost server in the "Server not running" section.

lms_embed() aborts at the first request that raises rlmstudio_model_mismatch, and no request goes out after it. A failed model-list lookup that raises rlmstudio_no_server, or an rlmstudio_api_error that would end the call as an embeddings failure, such as one with status 401, 403, or 404, ends the call in the same way. Once a request has succeeded, the condition carries a results field, a matrix or a list as the "Server not running" section describes. Any other rlmstudio_api_error or rlmstudio_bad_response from the lookup fails the inputs of that request alone, as the "API failure" section describes for a failed request. An error of any other class, such as an httr2_failure from a server that stops during the lookup, aborts the call unchanged.

---

lms_daemon_start *Start the LM Studio headless daemon*

Description

Launches the llmster daemon in the background via the CLI. This is required in headless environments (such as Linux servers) before loading models or starting the local server.

Usage

```
lms_daemon_start()
```

Details

If the CLI exits with a status other than 0, the function aborts. The message gives the exit code and quotes what the CLI wrote, after "The CLI said:". The quoted text is the stderr text, or the stdout text if stderr holds only whitespace and escape codes. A byte that is not valid UTF-8 shows as <xx>, its hex value. ANSI escape codes, such as color codes, cursor codes, and terminal links, are removed, and each run of whitespace becomes one space. A text longer than 1000 characters keeps at most its last 1000 characters, after "...".

Value

Invisibly returns the CLI exit code, 0.

Desktop Users

On desktop operating systems (macOS and Windows), running this command may actually launch the LM Studio desktop application to act as the backend engine. If the GUI is already open, this function will simply detect the active instance and return successfully. While safe to use, desktop users generally do not need to call this function and can just open the application manually.

See Also

LM Studio Headless Daemon (llmster)

Examples

```
## Not run:
lms_daemon_start()

## End(Not run)
```

---

lms_daemon_status *Check the global status of LM Studio*

Description

Displays the overall status of the LM Studio backend via the CLI, including loaded models and the server state. This function works regardless of whether the backend was started via the desktop GUI or the headless daemon.

Usage

```
lms_daemon_status()
```

Value

A character vector of the raw CLI output.

Examples

```
## Not run:
lms_daemon_status()

## End(Not run)
```

---

lms_daemon_stop *Stop the LM Studio headless daemon*

Description

Stops the `llmster` daemon via the CLI. Use this to clean up system resources when you are completely finished using LM Studio in headless mode.

Usage

```
lms_daemon_stop(force = FALSE)
```

Arguments

`force`	`TRUE` or `FALSE`. If `TRUE`, attempts to stop the local server before shutting down the daemon. The daemon cannot be stopped while the server is actively running. Defaults to `FALSE`. If no server is running, the message of `lms_server_stop()` says so. Any other value, `NULL` and `NA` included, aborts before the `lms` CLI runs.

Details

If the CLI exits with a status other than 0, the function reads what the CLI wrote. If the text says that the daemon is part of LM Studio, the function prints an info message and returns FALSE. If the text says that the daemon is not running, it prints an info message and returns TRUE. Letter case does not matter. Any other failure aborts. The message gives the exit code, quotes what the CLI wrote after "The CLI said:", and gives a hint about force = TRUE. The quoted text is the stderr text, or the stdout text if stderr holds only whitespace and escape codes. A byte that is not valid UTF-8 shows as <xx>, its hex value. ANSI escape codes, such as color codes, cursor codes, and terminal links, are removed, and each run of whitespace becomes one space. A text longer than 1000 characters keeps at most its last 1000 characters, after "...".

Value

Invisibly returns TRUE if the daemon stopped or was not running, and FALSE if the LM Studio GUI manages it.

Desktop Users

If the daemon is currently being managed by the LM Studio desktop application, the CLI does not stop it. The CLI intentionally prevents programmatic shutdowns of the GUI to avoid disrupting visual sessions. This function then returns FALSE, and the daemon keeps running. In this scenario, you must close the desktop application manually.

Examples

```
## Not run:
lms_daemon_stop(force = TRUE)

## End(Not run)
```

---

lms_download *Download a model via REST API*

Description

Download a model via REST API

Usage

```
lms_download(
  model,
  quantization = NULL,
  host = "http://localhost:1234",
  ...,
  token = NULL
)
```

Arguments

model	Character. The model to download. Accepts model catalog identifiers (e.g., "openai/gpt-oss-20b") and exact Hugging Face links. Must be one name, given as a single string. The string must be valid in its declared encoding and not marked "bytes". A class, names, and the S4 bit are removed before the name is sent.
quantization	Character. Optional. Quantization level of the model to download (e.g., "Q4_K_M"). Only supported for Hugging Face links.
host	Character. The host address of the local server. Defaults to "http://localhost:1234".
...	Additional arguments passed to the request.
token	Character or NULL. An API token for a server that requires authentication. NULL reads the rlmstudio.token option and then the RLMSTUDIO_API_TOKEN environment variable. See rlmstudio_token.

Value

A character string containing the download job_id, or "already_downloaded", invisibly, if the model is already downloaded. The call aborts with rlmstudio_bad_response when the reply's status is "failed". It also aborts when the status is not "already_downloaded" and the reply holds no job_id string with a character that is not whitespace. It also aborts when the reply is not a JSON object whose status is a string.

Server not running

Functions that call the LM Studio REST API open a TCP connection to the hostname and port named in host before they send the request. A function that checks its own arguments does that first, so a bad model, job_id, input, inputs, messages, schema, ttl, previous_response_id, or batch_size, a bad type of list_models() or list_instances(), a bad TRUE or FALSE argument such as simplify, logprobs, quiet, force, or store, a store on the "openai" route of lms_chat() or lms_chat_batch(), a stream in the ... of a chat function, a logprobs in the ... of lms_chat_batch() or lms_chat_native(), a value in the ... of lms_chat_batch() that lms_chat() reads as an argument already given, an instructions or messages in the ... of lms_chat() or lms_chat_batch() on the route where lms_chat() sets it, or a name, id, or type string that is not valid in its encoding or is marked as bytes, aborts with an argument message and no condition class even when the server is down. A condition of class rlmstudio_no_server is raised when that connection cannot be opened. A refused connection raises it. So do an address the package cannot parse and a hostname that does not resolve. An address that neither accepts nor refuses the connection also raises it. That case waits for the operating system to give up, which can take a minute. If the hostname of host is localhost, in any letter case, or 127.0.0.1, start the server with lms_server_start() and the port of host. For the default host, that is lms_server_start(port = 1234). With no port, LM Studio uses the port that it used last. For any other host, start the server on the machine that host names, or give host the address that your server listens on. A host that does not parse gets this second hint too. The condition message gives the same hint for the host of the call.

The check reads the port and nothing else. Any process holding that port accepts the connection, so the condition is not raised even though no LM Studio server is there. The call then does not raise rlmstudio_no_server, and what it does depends on what answers. On a status-200 body that

does not parse as JSON, the chat functions, lms_embed(), list_models(), list_instances(), lms_load(), lms_download(), lms_download_status(), and lms_unload_all() raise rlmstudio_bad_response. lms_unload() does not read the body, so it can report success. A body that parses as JSON but has another shape can come back unchanged with simplify = FALSE. With simplify = TRUE, the chat functions and lms_embed() raise rlmstudio_bad_response for it. list_models() and list_instances() raise it for a model list with another shape, and so do lms_unload_all() and lms_load() without force = TRUE, which read that list. lms_chat_openai() and lms_chat_openresponses() raise it for such a model list too, with either setting of simplify, through the model lookup that a reply from another model starts. lms_chat() raises it through them. lms_embed() raises it in the same way for the inputs of that request. lms_load(), lms_download(), and lms_download_status() raise it for a reply of their own with another shape, such as {}. A process that does not answer in HTTP gives an httr2_failure error. Use lms_server_ready() for the stronger test: it asks the host for a model list and reports TRUE only for a model list that list_models() can read.

lms_chat_batch() checks the server once before its first input, and lms_chat() checks it again for each input. If that check finds the server gone during the batch, the batch aborts with rlmstudio_no_server, and no request goes out after that. The condition then carries a results field, a list as long as inputs. Its elements before the lost input hold the values that format = "list" returns for those inputs. The element of the lost input and every element after it are NULL. The check before the first input adds no results field. A connection that fails after the check passes, such as a server that stops during a request, raises an httr2_failure error instead. That error aborts the batch and carries no results field.

lms_embed() checks the server before each request, and a request carries at most batch_size inputs. If a check after the first request, or the check of a model-list lookup, finds the server gone, the call aborts with rlmstudio_no_server. Once a request has succeeded, the condition carries a results field. With simplify = TRUE, results is a matrix with NA in each row whose embedding did not arrive. With simplify = FALSE, it is a list with one element per batch, with NULL in the element of the request that ended the call and in every element after it.

API failure

A condition of class rlmstudio_api_error is raised when a REST call returns a response that the wrapper treats as a failure. The condition carries a status field, which holds the HTTP response status as an integer. It also carries a code field. The field holds the string at error.code of the response body, such as "model_not_found". It is NULL when the body does not parse, when error is not a JSON object, or when its code is not one string. A server_message field holds the error text as the server sent it: the error string of the body, or else the string at error.message. It is NULL when the body does not parse or holds neither string.

On LM Studio 0.4.26+4 with no model loaded, /v1/chat/completions, /v1/responses, and /v1/embeddings answered a model name that the server could not find with status 400, no code, and a server_message that starts with "No models loaded".

lms_chat_batch() aborts on it when its status is 401, 403, or 404, when its status is 400 and its code is "model_not_found", and when its status is 400 and its server_message starts with "No models loaded", in that letter case. No request goes out after that input. The condition then carries a results field that follows the rule for a lost server in the "Server not running" section: its elements before the failed input hold the values that format = "list" returns for those inputs, and the element of the failed input and every element after it are NULL. For any other status, the element of the failed input holds the condition, or NA where the result is text, and the batch warns once and goes on. See the details of lms_chat_batch().

lms_embed() follows the same rule for each request, except for the code "model_not_found". A 401, 403, or 404 aborts the call, and so does a 400 whose server_message starts with "No models loaded". Once a request has succeeded, the condition carries a results field, a matrix or a list as the "Server not running" section describes. Any other status, a 400 with the code "model_not_found" included, fails the inputs of that request alone, and the call warns once and goes on. If every request fails, the call aborts with the first condition. See the details of lms_embed().

Malformed response

A condition of class rlmstudio_bad_response is raised when the server answers with a status the wrapper accepts and a body the wrapper cannot read. It is raised where a wrapper checks the body before it reshapes it, rather than indexing straight into whatever arrived. Eleven functions raise it for a body they cannot read: lms_embed(), lms_chat(), lms_chat_native(), lms_chat_openresponses(), lms_chat_openai(), list_models(), list_instances(), lms_load(), lms_download(), lms_download_status(), and lms_unload_all(). lms_chat() raises it through the chat function it calls. lms_unload_all() raises it through list_models(), and so does lms_load() unless force = TRUE. lms_chat_batch() raises it only as rlmstudio_model_mismatch, which the "Reply from another model" section of lms_chat_openai() describes.

All eleven raise it for a status-200 body that does not parse as JSON, such as an HTML page from a proxy, JSON text that stops part way, or an empty body. In the functions that take simplify, the body is parsed before simplify is read, so the condition is raised whatever simplify is. The body is parsed by its content and not by its Content-Type header, so valid JSON under text/plain is read as JSON. The body is read as JSON text and nothing else. A body whose text is a URL or the path of a file does not parse, and the package does not fetch the URL or read the file. The message says that the body did not parse as JSON and that something other than LM Studio may be answering on the host. It does not hold the body text.

The condition carries a status field, which holds the HTTP response status as an integer. Today the status is always 200: each of these functions reads the body only after a 200, and reports every other status as an rlmstudio_api_error instead.

Malformed load or download reply

lms_load(), lms_download(), and lms_download_status() also raise it for a status-200 reply of their own with the wrong shape. Each reply must follow the rule of its function. Each field is read by its exact name. The rules check the type of a field and not its value, with four exceptions. The status of a load reply must be "loaded". A download reply whose status is "already_downloaded" needs no job_id. A download reply whose status is "failed" always aborts. The job_id of any other download reply must hold a character that is not whitespace.

1. A reply of lms_load() is a JSON object whose status is the string "loaded". With echo_load_config = TRUE, its load_config is also a JSON object.
2. A reply of lms_download() is a JSON object whose status is a string other than "failed". If the status is not "already_downloaded", its job_id is a string with a character that is not whitespace. For a "failed" status, the message says that LM Studio reports that the download failed. It names the reply's job_id if that is a string with a character that is not whitespace.
3. A reply of lms_download_status() is a JSON object whose job_id and status are strings. Its total_size_bytes, downloaded_bytes, and bytes_per_second are each a number, or absent, or null.

For the other faults, the message names the field that broke the rule, or it says that the body is not a JSON object. It also says that something other than LM Studio may be answering on the host.

See Also

LM Studio Download Model API

Examples

```
## Not run:
lms_server_start(port = 1234)

# Download a model by its HuggingFace identifier
job_id <- lms_download("google/gemma-3-1b")

# Download with a specific quantization level
lms_download("google/gemma-3-1b", quantization = "4bit")

## End(Not run)
```

---

lms_download_status *Get the status of a download job*

Description

Get the status of a download job

Usage

```
lms_download_status(job_id, host = "http://localhost:1234", token = NULL)
```

Arguments

job_id	Character. The unique identifier for the download job. Must be one id, given as a single string. The string must be valid in its declared encoding and not marked "bytes". A class, names, and the S4 bit are removed before the id is sent.
host	Character. The host address of the local server. Defaults to "http://localhost:1234".
token	Character or NULL. An API token for a server that requires authentication. NULL reads the rlmstudio.token option and then the RLMSTUDIO_API_TOKEN environment variable. See rlmstudio_token.

Value

An object of class lms_download_status containing the download status.

Server not running

Functions that call the LM Studio REST API open a TCP connection to the hostname and port named in `host` before they send the request. A function that checks its own arguments does that first, so a bad `model`, `job_id`, `input`, `inputs`, `messages`, `schema`, `ttl`, `previous_response_id`, or `batch_size`, a bad type of `list_models()` or `list_instances()`, a bad `TRUE` or `FALSE` argument such as `simplify`, `logprobs`, `quiet`, `force`, or `store`, a `store` on the `"openai"` route of `lms_chat()` or `lms_chat_batch()`, a `stream` in the `...` of a chat function, a `logprobs` in the `...` of `lms_chat_batch()` or `lms_chat_native()`, a value in the `...` of `lms_chat_batch()` that `lms_chat()` reads as an argument already given, an `instructions` or `messages` in the `...` of `lms_chat()` or `lms_chat_batch()` on the route where `lms_chat()` sets it, or a name, id, or type string that is not valid in its encoding or is marked as bytes, aborts with an argument message and no condition class even when the server is down. A condition of class `rlmstudio_no_server` is raised when that connection cannot be opened. A refused connection raises it. So do an address the package cannot parse and a hostname that does not resolve. An address that neither accepts nor refuses the connection also raises it. That case waits for the operating system to give up, which can take a minute. If the hostname of `host` is `localhost`, in any letter case, or `127.0.0.1`, start the server with `lms_server_start()` and the port of `host`. For the default `host`, that is `lms_server_start(port = 1234)`. With no `port`, LM Studio uses the port that it used last. For any other `host`, start the server on the machine that `host` names, or give `host` the address that your server listens on. A `host` that does not parse gets this second hint too. The condition message gives the same hint for the `host` of the call.

The check reads the port and nothing else. Any process holding that port accepts the connection, so the condition is not raised even though no LM Studio server is there. The call then does not raise `rlmstudio_no_server`, and what it does depends on what answers. On a status-200 body that does not parse as JSON, the chat functions, `lms_embed()`, `list_models()`, `list_instances()`, `lms_load()`, `lms_download()`, `lms_download_status()`, and `lms_unload_all()` raise `rlmstudio_bad_response`. `lms_unload()` does not read the body, so it can report success. A body that parses as JSON but has another shape can come back unchanged with `simplify = FALSE`. With `simplify = TRUE`, the chat functions and `lms_embed()` raise `rlmstudio_bad_response` for it. `list_models()` and `list_instances()` raise it for a model list with another shape, and so do `lms_unload_all()` and `lms_load()` without `force = TRUE`, which read that list. `lms_chat_openai()` and `lms_chat_openresponses()` raise it for such a model list too, with either setting of `simplify`, through the model lookup that a reply from another model starts. `lms_chat()` raises it through them. `lms_embed()` raises it in the same way for the inputs of that request. `lms_load()`, `lms_download()`, and `lms_download_status()` raise it for a reply of their own with another shape, such as `{}`. A process that does not answer in HTTP gives an `httr2_failure` error. Use `lms_server_ready()` for the stronger test: it asks the host for a model list and reports `TRUE` only for a model list that `list_models()` can read.

`lms_chat_batch()` checks the server once before its first input, and `lms_chat()` checks it again for each input. If that check finds the server gone during the batch, the batch aborts with `rlmstudio_no_server`, and no request goes out after that. The condition then carries a `results` field, a list as long as `inputs`. Its elements before the lost input hold the values that `format = "list"` returns for those inputs. The element of the lost input and every element after it are `NULL`. The check before the first input adds no `results` field. A connection that fails after the check passes, such as a server that stops during a request, raises an `httr2_failure` error instead. That error aborts the batch and carries no `results` field.

`lms_embed()` checks the server before each request, and a request carries at most `batch_size` inputs. If a check after the first request, or the check of a model-list lookup, finds the server gone,

the call aborts with rlmstudio_no_server. Once a request has succeeded, the condition carries a results field. With simplify = TRUE, results is a matrix with NA in each row whose embedding did not arrive. With simplify = FALSE, it is a list with one element per batch, with NULL in the element of the request that ended the call and in every element after it.

API failure

A condition of class rlmstudio_api_error is raised when a REST call returns a response that the wrapper treats as a failure. The condition carries a status field, which holds the HTTP response status as an integer. It also carries a code field. The field holds the string at error.code of the response body, such as "model_not_found". It is NULL when the body does not parse, when error is not a JSON object, or when its code is not one string. A server_message field holds the error text as the server sent it: the error string of the body, or else the string at error.message. It is NULL when the body does not parse or holds neither string.

On LM Studio 0.4.26+4 with no model loaded, /v1/chat/completions, /v1/responses, and /v1/embeddings answered a model name that the server could not find with status 400, no code, and a server_message that starts with "No models loaded".

lms_chat_batch() aborts on it when its status is 401, 403, or 404, when its status is 400 and its code is "model_not_found", and when its status is 400 and its server_message starts with "No models loaded", in that letter case. No request goes out after that input. The condition then carries a results field that follows the rule for a lost server in the "Server not running" section: its elements before the failed input hold the values that format = "list" returns for those inputs, and the element of the failed input and every element after it are NULL. For any other status, the element of the failed input holds the condition, or NA where the result is text, and the batch warns once and goes on. See the details of lms_chat_batch().

lms_embed() follows the same rule for each request, except for the code "model_not_found". A 401, 403, or 404 aborts the call, and so does a 400 whose server_message starts with "No models loaded". Once a request has succeeded, the condition carries a results field, a matrix or a list as the "Server not running" section describes. Any other status, a 400 with the code "model_not_found" included, fails the inputs of that request alone, and the call warns once and goes on. If every request fails, the call aborts with the first condition. See the details of lms_embed().

Malformed response

A condition of class rlmstudio_bad_response is raised when the server answers with a status the wrapper accepts and a body the wrapper cannot read. It is raised where a wrapper checks the body before it reshapes it, rather than indexing straight into whatever arrived. Eleven functions raise it for a body they cannot read: lms_embed(), lms_chat(), lms_chat_native(), lms_chat_openresponses(), lms_chat_openai(), list_models(), list_instances(), lms_load(), lms_download(), lms_download_status(), and lms_unload_all(). lms_chat() raises it through the chat function it calls. lms_unload_all() raises it through list_models(), and so does lms_load() unless force = TRUE. lms_chat_batch() raises it only as rlmstudio_model_mismatch, which the "Reply from another model" section of lms_chat_openai() describes.

All eleven raise it for a status-200 body that does not parse as JSON, such as an HTML page from a proxy, JSON text that stops part way, or an empty body. In the functions that take simplify, the body is parsed before simplify is read, so the condition is raised whatever simplify is. The body is parsed by its content and not by its Content-Type header, so valid JSON under text/plain is read as JSON. The body is read as JSON text and nothing else. A body whose text is a URL

or the path of a file does not parse, and the package does not fetch the URL or read the file. The message says that the body did not parse as JSON and that something other than LM Studio may be answering on the host. It does not hold the body text.

The condition carries a `status` field, which holds the HTTP response status as an integer. Today the status is always 200: each of these functions reads the body only after a 200, and reports every other status as an `rlmstudio_api_error` instead.

Malformed load or download reply

`lms_load()`, `lms_download()`, and `lms_download_status()` also raise it for a status-200 reply of their own with the wrong shape. Each reply must follow the rule of its function. Each field is read by its exact name. The rules check the type of a field and not its value, with four exceptions. The `status` of a load reply must be `"loaded"`. A download reply whose `status` is `"already_downloaded"` needs no `job_id`. A download reply whose `status` is `"failed"` always aborts. The `job_id` of any other download reply must hold a character that is not whitespace.

1. A reply of `lms_load()` is a JSON object whose `status` is the string `"loaded"`. With `echo_load_config = TRUE`, its `load_config` is also a JSON object.
2. A reply of `lms_download()` is a JSON object whose `status` is a string other than `"failed"`. If the status is not `"already_downloaded"`, its `job_id` is a string with a character that is not whitespace. For a `"failed"` status, the message says that LM Studio reports that the download failed. It names the reply's `job_id` if that is a string with a character that is not whitespace.
3. A reply of `lms_download_status()` is a JSON object whose `job_id` and `status` are strings. Its `total_size_bytes`, `downloaded_bytes`, and `bytes_per_second` are each a number, or absent, or `null`.

For the other faults, the message names the field that broke the rule, or it says that the body is not a JSON object. It also says that something other than LM Studio may be answering on the host.

See Also

LM Studio Download Status API

Examples

```
## Not run:
lms_server_start(port = 1234)

job_id <- lms_download("google/gemma-3-1b")
status <- lms_download_status(job_id)
print(status)

## End(Not run)
```

`lms_embed` *Turn Text into Embedding Vectors*

Description

Sends one or more texts to an embedding model and returns the vector that the model produced for each one. The texts go out in batches of at most `batch_size`, one request per batch, in the order given. A reply that names another model can add one model-list request, as the "Reply from another model" section describes.

Usage

```
lms_embed(
  model,
  input,
  host = "http://localhost:1234",
  simplify = TRUE,
  ...,
  ttl = NULL,
  token = NULL,
  batch_size = 100,
  quiet = NULL
)
```

Arguments

`model`	Character. The loaded embedding model name. Must be one name, given as a single string. The string must be valid in its declared encoding and not marked `"bytes"`. A class, names, and the S4 bit are removed before the name is sent.
`input`	Character. The texts to embed. A vector of length `n` returns `n` embeddings, in the order given. Must hold at least one value and no missing values.
`host`	Character. Server URL.
`simplify`	`TRUE` or `FALSE`. If `TRUE`, the default, returns a numeric matrix with one row per input. `FALSE` returns a list with one element per request: the parsed response body, unchanged, or the condition of a request that failed. Any other value, `NULL` and `NA` included, aborts before the check for a running server.
`...`	Additional fields for the request body. LM Studio ignores a field it does not recognize, and two OpenAI fields are worth naming for that reason: LM Studio ignores `dimensions`, so asking for a narrower vector has no effect. `encoding_format = "base64"` is untested against LM Studio: a server that honors it returns embeddings this function cannot read, and the default `simplify = TRUE` path then aborts.
`ttl`	A whole number of seconds from 1 to `.Machine$integer.max`, or `NULL` to leave it out. It is how long the model stays loaded with no request. It has an effect only on a model that this request loads. The server loads a model that is not

loaded yet when its just-in-time loading setting is on. A model that is already loaded keeps its idle time.

`token` Character or `NULL`. An API token for a server that requires authentication. `NULL` reads the `rlmstudio.token` option and then the `RLMSTUDIO_API_TOKEN` environment variable. See rlmstudio_token.

`batch_size` A whole number from 1 to `.Machine$integer.max`. The most texts that one request carries. The default is 100. A value at least as large as `length(input)` sends every text in one request.

`quiet` `TRUE`, `FALSE`, or `NULL`, the default. `NULL` follows the `rlmstudio.quiet` option. `TRUE` starts no progress bar, and `FALSE` starts one, also when the option is `TRUE`. cli draws a started bar only after a delay, two seconds by default. A bar starts only when the call sends more than one request. Any other value, `NA` included, aborts before the check for a running server. `quiet` does not suppress the warning about failed inputs.

Details

Before each request after the first, the function checks again that the server is running.

A request that fails with an `rlmstudio_bad_response` other than an `rlmstudio_model_mismatch`, or with an `rlmstudio_api_error` that does not end the call as described below, fails the inputs it carried alone. Their rows hold `NA`, or their element of the `simplify = FALSE` list holds the condition without its backtrace. The call goes on to the next request and then gives one warning that names the count and the positions of the failed inputs. That warning shows even with `quiet = TRUE`. If every request fails, the call aborts with the condition of the first failed request, and no warning is given.

Four faults end the call at once, and no request goes out after them: a server that the check before a request, or the model-list lookup of the reply check, finds gone; an `rlmstudio_api_error` with `status` 401, 403, or 404; an `rlmstudio_api_error` with `status` 400 whose `server_message` starts with "No models loaded", which LM Studio sent for an unknown name with no model loaded; and an `rlmstudio_model_mismatch`, a reply from a model other than the one asked for. The "API failure" and "Reply from another model" sections below describe the last two. A fault that ends the call gives no warning about the inputs that failed before it. With `simplify = TRUE`, a fifth fault does the same: a request whose embeddings have another number of dimensions than those of an earlier request. It aborts with `rlmstudio_bad_response`. With `simplify = FALSE`, the bodies do not go through the matrix checks, so bodies of two widths are returned with no abort. Each abort after a request that succeeded carries a `results` field. With `simplify = TRUE`, it is the matrix so far, with `NA` in each row whose embedding did not arrive. With `simplify = FALSE`, it is the list so far, with `NULL` in the element of the request that ended the call and in every element after it. An abort before any request succeeded carries no `results` field. An error of any other class aborts the call unchanged.

On LM Studio 0.4.25+1, the server embeds only the first tokens of each text, up to the context length of the loaded model instance. For a text longer than that, it still returns a vector, with no error or warning, and the vector covers only the start of the text. The `usage` field of the reply reported 0 tokens on that version, so the count does not show the cut. The `loaded_instances` column of `list_models(detailed = TRUE)` gives the `context_length` of each loaded instance. To embed all of a long text, split it into pieces before the call. Or unload the model with `lms_unload()` and load it again with a larger `context_length` through `lms_load()`, up to the `max_context_length`

column of `list_models(detailed = TRUE)`. For a model that is already loaded, `lms_load()` does not load it again.

Value

If `simplify = FALSE`, a list with one element per request, in request order. Each element is the parsed JSON body of that request, or the condition of a request that failed. A call with one request returns a list of one. Otherwise, a double matrix with one row per input text and one column per embedding dimension. The row at position `i` holds the embedding that the response reported for the input at position `i`, or `NA` for an input whose request failed. The matrix carries no row or column names.

Server not running

Functions that call the LM Studio REST API open a TCP connection to the hostname and port named in `host` before they send the request. A function that checks its own arguments does that first, so a bad `model`, `job_id`, `input`, `inputs`, `messages`, `schema`, `ttl`, `previous_response_id`, or `batch_size`, a bad `type` of `list_models()` or `list_instances()`, a bad `TRUE` or `FALSE` argument such as `simplify`, `logprobs`, `quiet`, `force`, or `store`, a `store` on the `"openai"` route of `lms_chat()` or `lms_chat_batch()`, a `stream` in the `...` of a chat function, a `logprobs` in the `...` of `lms_chat_batch()` or `lms_chat_native()`, a value in the `...` of `lms_chat_batch()` that `lms_chat()` reads as an argument already given, an `instructions` or `messages` in the `...` of `lms_chat()` or `lms_chat_batch()` on the route where `lms_chat()` sets it, or a name, id, or `type` string that is not valid in its encoding or is marked as bytes, aborts with an argument message and no condition class even when the server is down. A condition of class `rlmstudio_no_server` is raised when that connection cannot be opened. A refused connection raises it. So do an address the package cannot parse and a hostname that does not resolve. An address that neither accepts nor refuses the connection also raises it. That case waits for the operating system to give up, which can take a minute. If the hostname of `host` is `localhost`, in any letter case, or `127.0.0.1`, start the server with `lms_server_start()` and the port of `host`. For the default `host`, that is `lms_server_start(port = 1234)`. With no `port`, LM Studio uses the port that it used last. For any other `host`, start the server on the machine that `host` names, or give `host` the address that your server listens on. A `host` that does not parse gets this second hint too. The condition message gives the same hint for the `host` of the call.

The check reads the port and nothing else. Any process holding that port accepts the connection, so the condition is not raised even though no LM Studio server is there. The call then does not raise `rlmstudio_no_server`, and what it does depends on what answers. On a status-200 body that does not parse as JSON, the chat functions, `lms_embed()`, `list_models()`, `list_instances()`, `lms_load()`, `lms_download()`, `lms_download_status()`, and `lms_unload_all()` raise `rlmstudio_bad_response`. `lms_unload()` does not read the body, so it can report success. A body that parses as JSON but has another shape can come back unchanged with `simplify = FALSE`. With `simplify = TRUE`, the chat functions and `lms_embed()` raise `rlmstudio_bad_response` for it. `list_models()` and `list_instances()` raise it for a model list with another shape, and so do `lms_unload_all()` and `lms_load()` without `force = TRUE`, which read that list. `lms_chat_openai()` and `lms_chat_openresponses()` raise it for such a model list too, with either setting of `simplify`, through the model lookup that a reply from another model starts. `lms_chat()` raises it through them. `lms_embed()` raises it in the same way for the inputs of that request. `lms_load()`, `lms_download()`, and `lms_download_status()` raise it for a reply of their own with another shape, such as `{}`. A process that does not answer in

HTTP gives an `httr2_failure` error. Use `lms_server_ready()` for the stronger test: it asks the host for a model list and reports `TRUE` only for a model list that `list_models()` can read.

`lms_chat_batch()` checks the server once before its first input, and `lms_chat()` checks it again for each input. If that check finds the server gone during the batch, the batch aborts with `rlmstudio_no_server`, and no request goes out after that. The condition then carries a `results` field, a list as long as `inputs`. Its elements before the lost input hold the values that `format = "list"` returns for those inputs. The element of the lost input and every element after it are `NULL`. The check before the first input adds no `results` field. A connection that fails after the check passes, such as a server that stops during a request, raises an `httr2_failure` error instead. That error aborts the batch and carries no `results` field.

`lms_embed()` checks the server before each request, and a request carries at most `batch_size` inputs. If a check after the first request, or the check of a model-list lookup, finds the server gone, the call aborts with `rlmstudio_no_server`. Once a request has succeeded, the condition carries a `results` field. With `simplify = TRUE`, `results` is a matrix with `NA` in each row whose embedding did not arrive. With `simplify = FALSE`, it is a list with one element per batch, with `NULL` in the element of the request that ended the call and in every element after it.

API failure

A condition of class `rlmstudio_api_error` is raised when a REST call returns a response that the wrapper treats as a failure. The condition carries a `status` field, which holds the HTTP response status as an integer. It also carries a `code` field. The field holds the string at `error.code` of the response body, such as `"model_not_found"`. It is `NULL` when the body does not parse, when `error` is not a JSON object, or when its `code` is not one string. A `server_message` field holds the error text as the server sent it: the `error` string of the body, or else the string at `error.message`. It is `NULL` when the body does not parse or holds neither string.

On LM Studio 0.4.26+4 with no model loaded, `/v1/chat/completions`, `/v1/responses`, and `/v1/embeddings` answered a model name that the server could not find with status 400, no `code`, and a `server_message` that starts with "No models loaded".

`lms_chat_batch()` aborts on it when its `status` is 401, 403, or 404, when its `status` is 400 and its `code` is `"model_not_found"`, and when its `status` is 400 and its `server_message` starts with "No models loaded", in that letter case. No request goes out after that input. The condition then carries a `results` field that follows the rule for a lost server in the "Server not running" section: its elements before the failed input hold the values that `format = "list"` returns for those inputs, and the element of the failed input and every element after it are `NULL`. For any other status, the element of the failed input holds the condition, or `NA` where the result is text, and the batch warns once and goes on. See the details of `lms_chat_batch()`.

`lms_embed()` follows the same rule for each request, except for the code `"model_not_found"`. A 401, 403, or 404 aborts the call, and so does a 400 whose `server_message` starts with "No models loaded". Once a request has succeeded, the condition carries a `results` field, a matrix or a list as the "Server not running" section describes. Any other status, a 400 with the code `"model_not_found"` included, fails the inputs of that request alone, and the call warns once and goes on. If every request fails, the call aborts with the first condition. See the details of `lms_embed()`.

Malformed response

A condition of class `rlmstudio_bad_response` is raised when the server answers with a status the wrapper accepts and a body the wrapper cannot read. It is raised where a wrapper checks the body

before it reshapes it, rather than indexing straight into whatever arrived. Eleven functions raise it for a body they cannot read: `lms_embed()`, `lms_chat()`, `lms_chat_native()`, `lms_chat_openresponses()`, `lms_chat_openai()`, `list_models()`, `list_instances()`, `lms_load()`, `lms_download()`, `lms_download_status()`, and `lms_unload_all()`. `lms_chat()` raises it through the chat function it calls. `lms_unload_all()` raises it through `list_models()`, and so does `lms_load()` unless `force = TRUE`. `lms_chat_batch()` raises it only as `rlmstudio_model_mismatch`, which the "Reply from another model" section of `lms_chat_openai()` describes.

All eleven raise it for a status-200 body that does not parse as JSON, such as an HTML page from a proxy, JSON text that stops part way, or an empty body. In the functions that take `simplify`, the body is parsed before `simplify` is read, so the condition is raised whatever `simplify` is. The body is parsed by its content and not by its `Content-Type` header, so valid JSON under `text/plain` is read as JSON. The body is read as JSON text and nothing else. A body whose text is a URL or the path of a file does not parse, and the package does not fetch the URL or read the file. The message says that the body did not parse as JSON and that something other than LM Studio may be answering on the host. It does not hold the body text.

The condition carries a `status` field, which holds the HTTP response status as an integer. Today the status is always 200: each of these functions reads the body only after a 200, and reports every other status as an `rlmstudio_api_error` instead.

Malformed embeddings

`lms_embed()` raises it on an embeddings block it cannot trust. The vectors it returns are placed by the index that the response reports, so a block with a missing, repeated, or out-of-range index would otherwise pair a vector with the wrong text and give back a matrix that is silently wrong.

`lms_embed()` reads each request on its own. A bad body, of either kind that the "Malformed response" section and this section describe, fails the inputs of that request alone, and the call warns once and goes on. Unless a stop that the details of `lms_embed()` list ends the call first, the call aborts with the condition only if every request fails, and then with the condition of the first. With `simplify = TRUE`, it also aborts with `rlmstudio_bad_response` for a request whose embeddings have another number of dimensions than those of an earlier request. That condition carries a `results` field, a matrix as the "Server not running" section describes. An `rlmstudio_model_mismatch` also ends the call, as the "Reply from another model" section says. See the details of `lms_embed()`.

The messages of `lms_embed()` name `simplify = FALSE`, which returns the body unchanged, with three exceptions. A body that did not parse as JSON is checked before that argument is read, so its message points at the host instead. The messages of `rlmstudio_model_mismatch` and of a failed model-list lookup do not name it, because the check runs with either setting of `simplify`.

Reply from another model

`lms_chat_openai()`, `lms_chat_openresponses()`, and `lms_embed()` read the `model` field of each status-200 reply, with either setting of `simplify`. On LM Studio 0.4.25+1 with one chat model loaded, both chat routes answered a model name that the server could not find with status 200 and a reply from the loaded model. With two chat models loaded, they answered with status 400 and the code `"model_not_found"`, as the "API failure" section describes. On LM Studio 0.4.26+4 with one embedding model loaded, `/v1/embeddings` answered a name that it could not find with status 200 and vectors from the loaded embedding model. The `model` field names the loaded instance that answered. Its id can differ from the key that the call asked for.

If the model field equals the name that the call sent, the reply is accepted. If it differs, the call sends one request for the model list to api/v1/models on the same host, with the token of the call, and prints no message. The reply is accepted if a model whose key equals the asked name in any letter case has a loaded instance whose id is the model field. Otherwise the call aborts with a condition of class rlmstudio_model_mismatch. That condition is also an rlmstudio_bad_response, and its status is 200. Its model field holds the asked name, and its reply_model field holds the model field of the reply. The message names both. A condition from lms_chat_openai() also carries the content and finish_reason fields, both NULL.

Within one call of lms_chat_batch() or lms_embed(), a reply whose model field a lookup of that call already accepted for the asked name and the same host is accepted with no other request. Each such call, and each call of lms_chat_openai() or lms_chat_openresponses() outside a batch, starts with no accepted ids.

The check runs before the reply is read, so it also runs with logprobs = TRUE, with a schema on lms_chat_openai(), and for a reply with no answer text. In lms_embed(), it runs before the vectors are read. A reply is not checked if its body is not a JSON object, if it has no model field, or if its model is not one string or holds only whitespace. A body that is not a JSON object is returned with simplify = FALSE, and with simplify = TRUE it raises the error that the "Malformed chat reply" section describes, or for lms_embed() the "Malformed embeddings" section. If the instance that answered is unloaded before the model-list request, the call aborts, also when that instance belongs to the asked model.

If the model-list request fails, the check raises the condition of that failure. The message opens with the label of the function, followed by "because the model-list lookup failed". A status other than 200 raises rlmstudio_api_error. A body that does not parse as JSON, or that breaks a rule of a model list in the "Malformed model list" section, raises rlmstudio_bad_response. A server that the port check before the request cannot reach raises rlmstudio_no_server. A condition from the lookup does not carry the content and finish_reason fields, also when lms_chat_openai() raises it.

lms_chat() runs the check through lms_chat_openai() and lms_chat_openresponses(). lms_chat_native() does not check the reply. On LM Studio 0.4.25+1, /api/v1/chat answered a name that it could not find with status 404, which raises rlmstudio_api_error.

lms_chat_batch() aborts at the first input that raises rlmstudio_model_mismatch, on the "openai" and "openresponses" routes. It also aborts at an rlmstudio_api_error with status 400 whose code is "model_not_found", and at one with status 400 whose server_message starts with "No models loaded". No request goes out after that input. In each case, the condition carries a results field that follows the rule for a lost server in the "Server not running" section.

lms_embed() aborts at the first request that raises rlmstudio_model_mismatch, and no request goes out after it. A failed model-list lookup that raises rlmstudio_no_server, or an rlmstudio_api_error that would end the call as an embeddings failure, such as one with status 401, 403, or 404, ends the call in the same way. Once a request has succeeded, the condition carries a results field, a matrix or a list as the "Server not running" section describes. Any other rlmstudio_api_error or rlmstudio_bad_response from the lookup fails the inputs of that request alone, as the "API failure" section describes for a failed request. An error of any other class, such as an httr2_failure from a server that stops during the lookup, aborts the call unchanged.

Examples

```
## Not run:
```

```
vectors <- lms_embed(
  model = "text-embedding-nomic-embed-text-v1.5",
  input = c("the first document", "the second document")
)
dim(vectors)

## End(Not run)
```

`lms_load` *Load a model via REST API*

Description

Load a model via REST API

Usage

```
lms_load(
  model,
  context_length = NULL,
  eval_batch_size = NULL,
  flash_attention = NULL,
  num_experts = NULL,
  offload_kv_cache_to_gpu = NULL,
  echo_load_config = FALSE,
  force = FALSE,
  host = "http://localhost:1234",
  ...,
  token = NULL
)
```

Arguments

`model`	Character. Unique identifier for the model to load. Must be one name, given as a single string. The string must be valid in its declared encoding and not marked `"bytes"`. A class, names, and the S4 bit are removed before the name is sent.
`context_length`	Integer. Maximum number of tokens that the model will consider. With `force = FALSE`, a value above the maximum in the model list gives a warning. See the "Long prompts" section.
`eval_batch_size`	Integer. Number of input tokens to process together in a single batch during evaluation.
`flash_attention`	`TRUE`, `FALSE`, or `NULL`. Whether to optimize attention computation. `NULL`, the default, leaves the field out of the request. Any other value, `NA` and `"true"` included, aborts before the check for a running server.
`num_experts`	Integer. Number of experts to use during inference for MoE models.

offload_kv_cache_to_gpu
: TRUE, FALSE, or NULL. Whether KV cache is offloaded to GPU memory. NULL, the default, leaves the field out of the request. Any other value, NA and "true" included, aborts before the check for a running server.

echo_load_config
: TRUE or FALSE. If TRUE, echoes the final load configuration in the response. Any other value, NULL and NA included, aborts before the check for a running server.

force
: TRUE or FALSE. If TRUE, bypasses the check for currently loaded models and requests a new instance from the server. Note that this does not overwrite or replace the existing model; it loads a second concurrent instance into VRAM. Defaults to FALSE. Any other value, NULL and NA included, aborts before the check for a running server.

host
: Character. The host address of the local server. Defaults to "http://localhost:1234".

...
: Additional arguments passed to the API request body (useful for future API parameters).

token
: Character or NULL. An API token for a server that requires authentication. NULL reads the rlmstudio.token option and then the RLMSTUDIO_API_TOKEN environment variable. See rlmstudio_token.

Value

Invisibly returns a character string of the loaded model identifier upon success, also when the model was already loaded. It is a plain string, with no class, names, or S4 bit. If echo_load_config = TRUE and this call loads the model, it instead invisibly returns a list containing the model's detailed load configuration. If echo_load_config = TRUE and the model is already loaded, with force = FALSE, the call returns the model key and gives a warning of class rlmstudio_no_load_config. The rlmstudio.quiet option does not hide it.

Long prompts

A prompt longer than the context length of the loaded model fails. On LM Studio 0.4.25+1, with google/gemma-3-1b loaded at a context_length of 512, the /v1/responses and /api/v1/chat routes answered a longer prompt with status 500. The /v1/chat/completions route answered it with status 400. Each message began "The number of tokens to keep from the initial prompt is greater than the context length". The chat functions raise such a reply as an rlmstudio_api_error, and the message holds the server text. lms_chat_batch() fails that input alone and goes on to the next input. With format = "list", the element of that input holds the condition. With format = "vector", it holds NA.

To fit a longer prompt, load the model with a larger context_length in lms_load(). When the server reports it, the max_context_length column of list_models(detailed = TRUE) gives the largest context length that the model list reports for each model. If context_length is larger, the server loads the model with the asked value and gives no message. On LM Studio 0.4.25+1, it loaded google/gemma-3-1b at 65536 tokens, above its maximum of 32768. So lms_load() gives a warning of class rlmstudio_context_above_max that names both numbers, and then sends the load. The rlmstudio.quiet option does not hide it. The warning needs the model list, so it comes only with force = FALSE, and only when the list has a maximum for the model and the model is not loaded yet.

On LM Studio 0.4.25+1, the load endpoint answered a `rope_frequency_scale` field in `...` with status 400 and the code `"unrecognized_keys"`. So `lms_load()` cannot set RoPE scaling through that field. RoPE scaling stretches the position encoding of a model past its trained length.

To see how many tokens a prompt took, use `lms_chat_batch(format = "data.frame")`. Its `input_tokens` column holds the prompt token count that the server reports for each reply, on every route.

Server not running

Functions that call the LM Studio REST API open a TCP connection to the hostname and port named in `host` before they send the request. A function that checks its own arguments does that first, so a bad `model`, `job_id`, `input`, `inputs`, `messages`, `schema`, `ttl`, `previous_response_id`, or `batch_size`, a bad type of `list_models()` or `list_instances()`, a bad `TRUE` or `FALSE` argument such as `simplify`, `logprobs`, `quiet`, `force`, or `store`, a `store` on the `"openai"` route of `lms_chat()` or `lms_chat_batch()`, a `stream` in the `...` of a chat function, a `logprobs` in the `...` of `lms_chat_batch()` or `lms_chat_native()`, a value in the `...` of `lms_chat_batch()` that `lms_chat()` reads as an argument already given, an `instructions` or `messages` in the `...` of `lms_chat()` or `lms_chat_batch()` on the route where `lms_chat()` sets it, or a name, id, or type string that is not valid in its encoding or is marked as bytes, aborts with an argument message and no condition class even when the server is down. A condition of class `rlmstudio_no_server` is raised when that connection cannot be opened. A refused connection raises it. So do an address the package cannot parse and a hostname that does not resolve. An address that neither accepts nor refuses the connection also raises it. That case waits for the operating system to give up, which can take a minute. If the hostname of `host` is `localhost`, in any letter case, or `127.0.0.1`, start the server with `lms_server_start()` and the port of `host`. For the default `host`, that is `lms_server_start(port = 1234)`. With no `port`, LM Studio uses the port that it used last. For any other `host`, start the server on the machine that `host` names, or give `host` the address that your server listens on. A `host` that does not parse gets this second hint too. The condition message gives the same hint for the `host` of the call.

The check reads the port and nothing else. Any process holding that port accepts the connection, so the condition is not raised even though no LM Studio server is there. The call then does not raise `rlmstudio_no_server`, and what it does depends on what answers. On a status-200 body that does not parse as JSON, the chat functions, `lms_embed()`, `list_models()`, `list_instances()`, `lms_load()`, `lms_download()`, `lms_download_status()`, and `lms_unload_all()` raise `rlmstudio_bad_response`. `lms_unload()` does not read the body, so it can report success. A body that parses as JSON but has another shape can come back unchanged with `simplify = FALSE`. With `simplify = TRUE`, the chat functions and `lms_embed()` raise `rlmstudio_bad_response` for it. `list_models()` and `list_instances()` raise it for a model list with another shape, and so do `lms_unload_all()` and `lms_load()` without `force = TRUE`, which read that list. `lms_chat_openai()` and `lms_chat_openresponses()` raise it for such a model list too, with either setting of `simplify`, through the model lookup that a reply from another model starts. `lms_chat()` raises it through them. `lms_embed()` raises it in the same way for the inputs of that request. `lms_load()`, `lms_download()`, and `lms_download_status()` raise it for a reply of their own with another shape, such as `{}`. A process that does not answer in HTTP gives an `httr2_failure` error. Use `lms_server_ready()` for the stronger test: it asks the host for a model list and reports `TRUE` only for a model list that `list_models()` can read.

`lms_chat_batch()` checks the server once before its first input, and `lms_chat()` checks it again for each input. If that check finds the server gone during the batch, the batch aborts with `rlmstudio_no_server`, and no request goes out after that. The condition then carries a `results` field, a list as long as

inputs. Its elements before the lost input hold the values that format = "list" returns for those inputs. The element of the lost input and every element after it are NULL. The check before the first input adds no results field. A connection that fails after the check passes, such as a server that stops during a request, raises an httr2_failure error instead. That error aborts the batch and carries no results field.

lms_embed() checks the server before each request, and a request carries at most batch_size inputs. If a check after the first request, or the check of a model-list lookup, finds the server gone, the call aborts with rlmstudio_no_server. Once a request has succeeded, the condition carries a results field. With simplify = TRUE, results is a matrix with NA in each row whose embedding did not arrive. With simplify = FALSE, it is a list with one element per batch, with NULL in the element of the request that ended the call and in every element after it.

API failure

A condition of class rlmstudio_api_error is raised when a REST call returns a response that the wrapper treats as a failure. The condition carries a status field, which holds the HTTP response status as an integer. It also carries a code field. The field holds the string at error.code of the response body, such as "model_not_found". It is NULL when the body does not parse, when error is not a JSON object, or when its code is not one string. A server_message field holds the error text as the server sent it: the error string of the body, or else the string at error.message. It is NULL when the body does not parse or holds neither string.

On LM Studio 0.4.26+4 with no model loaded, /v1/chat/completions, /v1/responses, and /v1/embeddings answered a model name that the server could not find with status 400, no code, and a server_message that starts with "No models loaded".

lms_chat_batch() aborts on it when its status is 401, 403, or 404, when its status is 400 and its code is "model_not_found", and when its status is 400 and its server_message starts with "No models loaded", in that letter case. No request goes out after that input. The condition then carries a results field that follows the rule for a lost server in the "Server not running" section: its elements before the failed input hold the values that format = "list" returns for those inputs, and the element of the failed input and every element after it are NULL. For any other status, the element of the failed input holds the condition, or NA where the result is text, and the batch warns once and goes on. See the details of lms_chat_batch().

lms_embed() follows the same rule for each request, except for the code "model_not_found". A 401, 403, or 404 aborts the call, and so does a 400 whose server_message starts with "No models loaded". Once a request has succeeded, the condition carries a results field, a matrix or a list as the "Server not running" section describes. Any other status, a 400 with the code "model_not_found" included, fails the inputs of that request alone, and the call warns once and goes on. If every request fails, the call aborts with the first condition. See the details of lms_embed().

Malformed response

A condition of class rlmstudio_bad_response is raised when the server answers with a status the wrapper accepts and a body the wrapper cannot read. It is raised where a wrapper checks the body before it reshapes it, rather than indexing straight into whatever arrived. Eleven functions raise it for a body they cannot read: lms_embed(), lms_chat(), lms_chat_native(), lms_chat_openresponses(), lms_chat_openai(), list_models(), list_instances(), lms_load(), lms_download(), lms_download_status(), and lms_unload_all(). lms_chat() raises it through the chat function it calls. lms_unload_all() raises it through list_models(), and so does lms_load() unless force = TRUE. lms_chat_batch()

raises it only as `rlmstudio_model_mismatch`, which the "Reply from another model" section of `lms_chat_openai()` describes.

All eleven raise it for a status-200 body that does not parse as JSON, such as an HTML page from a proxy, JSON text that stops part way, or an empty body. In the functions that take `simplify`, the body is parsed before `simplify` is read, so the condition is raised whatever `simplify` is. The body is parsed by its content and not by its `Content-Type` header, so valid JSON under `text/plain` is read as JSON. The body is read as JSON text and nothing else. A body whose text is a URL or the path of a file does not parse, and the package does not fetch the URL or read the file. The message says that the body did not parse as JSON and that something other than LM Studio may be answering on the host. It does not hold the body text.

The condition carries a `status` field, which holds the HTTP response status as an integer. Today the status is always 200: each of these functions reads the body only after a 200, and reports every other status as an `rlmstudio_api_error` instead.

Malformed model list

`list_models()` also raises it for a status-200 model list with the wrong shape. `lms_unload_all()` and `lms_load()` without `force = TRUE` raise it through `list_models()`. `lms_chat_openai()`, `lms_chat_openresponses()`, and `lms_embed()` apply these rules to the model list of their model lookup, as the "Reply from another model" section of `lms_chat_openai()` describes. A model list must follow four rules. Each field is read by its exact name, so a field named `keyX` does not stand in for `key`.

1. The body is a JSON object whose `models` field is an array. The array can be empty.
2. Each entry of `models` is a JSON object. Its `type` and `key` are strings, and its `loaded_instances` is an array.
3. The `size_bytes` of an entry is a number, or absent, or `null`. So is its `max_context_length`.
4. Each entry of `loaded_instances` is a JSON object whose `id` is a string with a character that is not whitespace.

The rules are checked before the `type` and `loaded` filters, so an entry that the filters drop can still raise the condition. The message names the field or entry that broke a rule. `lms_server_ready()` applies the same rules and returns `FALSE` for a body that breaks one.

`list_instances()` applies the four rules too, and three more. They cover only an entry whose `type` is in its `type` argument and that has at least one loaded instance. The `display_name` of such an entry is a string, or absent, or `null`. The `remaining_ttl_seconds` of each of its instances is a number, or absent, or `null`. The `config` of each of its instances is a JSON object, or absent, or `null`. `list_models()` and `lms_server_ready()` do not apply these three rules.

Malformed load or download reply

`lms_load()`, `lms_download()`, and `lms_download_status()` also raise it for a status-200 reply of their own with the wrong shape. Each reply must follow the rule of its function. Each field is read by its exact name. The rules check the type of a field and not its value, with four exceptions. The `status` of a load reply must be `"loaded"`. A download reply whose `status` is `"already_downloaded"` needs no `job_id`. A download reply whose `status` is `"failed"` always aborts. The `job_id` of any other download reply must hold a character that is not whitespace.

1. A reply of lms_load() is a JSON object whose status is the string "loaded". With echo_load_config = TRUE, its load_config is also a JSON object.
2. A reply of lms_download() is a JSON object whose status is a string other than "failed". If the status is not "already_downloaded", its job_id is a string with a character that is not whitespace. For a "failed" status, the message says that LM Studio reports that the download failed. It names the reply's job_id if that is a string with a character that is not whitespace.
3. A reply of lms_download_status() is a JSON object whose job_id and status are strings. Its total_size_bytes, downloaded_bytes, and bytes_per_second are each a number, or absent, or null.

For the other faults, the message names the field that broke the rule, or it says that the body is not a JSON object. It also says that something other than LM Studio may be answering on the host.

See Also

LM Studio Load Model API

Examples

```
## Not run:
lms_server_start(port = 1234)
lms_download("google/gemma-3-1b")

# Load a model with default settings
lms_load("google/gemma-3-1b")

# Load a model with custom context length and flash attention enabled
lms_load("google/gemma-3-1b", context_length = 8192, flash_attention = TRUE)

## End(Not run)
```

---

lms_path *Get the absolute path to the LMS executable*

Description

Locates the LM Studio CLI (lms) on your system. It checks the RLMSTUDIO_LMS_PATH environment variable first, then the system PATH, and finally common installation directories.

Usage

```
lms_path()
```

Value

A character string specifying the absolute file path to the LM Studio executable (lms) on the user's system.

Examples

```
## Not run:
lms_path()

## End(Not run)
```

`lms_score_expected` *Calculate Expected Scores and Uncertainty from Logprobs*

Description

Takes a logprobs dataframe (from an `lms_chat_result`) and calculates the weighted average score, normalized probabilities, and uncertainty metrics for the first step of the reply, where the rating is.

Usage

```
lms_score_expected(lp_df, scale = 1:5)
```

Arguments

`lp_df`	A dataframe of logprobs (e.g., `x$logprobs`).
`scale`	Numeric vector. The valid labels (e.g., `1:5`).

Details

The function reads the rows of the first step only. If `lp_df` has a `step` column and its first value is not `NA`, these are the rows whose `step` equals that value, wherever they sit in the frame. A later step with the same token as the first step does not count. Otherwise, these are the first run of consecutive rows whose `step_token` is identical to that of the first row, with `NA` equal to `NA`. In that case, with no `step` column or a first `step` of `NA`, two adjacent steps with the same token count as one step. Both rules start from the first row, so keep the rows in reply order: a sorted frame can start with a later step.

Of those rows, the function keeps the candidates whose token, read as a number, is in `scale`. Candidates that give the same label, such as `"3"`, `" 3"`, and `"3.0"`, are summed into one label. The probabilities of the labels are then scaled to sum to 1.

Value

A named list containing three numeric elements (`expected_value`, `weighted_sd`, `entropy`) and a `data.frame` named `probabilities` with columns `label` and `prob`. `probabilities` has one row per label, in the order in which each label first appears, and `entropy` is computed over these rows. Returns `NULL` if `lp_df` is `NULL` or has no rows. Aborts if no candidate of the first step is in `scale`.

Examples

```
# Create a sample logprobs dataframe representing a model's generation step
mock_logprobs <- data.frame(
  step_token = rep("4", 3),
  step_logprob = rep(0, 3),
  candidate_token = c("4", "5", "3"),
  candidate_logprob = c(-0.105, -2.302, -3.506),
  step = rep(1L, 3),
  stringsAsFactors = FALSE
)

# Calculate the expected score and uncertainty metrics
lms_score_expected(mock_logprobs, scale = 1:5)
```

---

lms_server_ready *Check whether a host answers as a usable LM Studio server*

Description

Sends one GET request to the model list endpoint at `host` and reports whether the answer came from an LM Studio server that this package can use. The answer is `TRUE` only when the request returns HTTP status 200 and the response body is a model list that `list_models()` can read. The two functions apply the same shape rules, which the "Malformed model list" section of `list_models()` states. An empty list counts, so a server that lists no models gives `TRUE`.

Usage

```
lms_server_ready(host = "http://localhost:1234", timeout = 2, token = NULL)
```

Arguments

`host`	Character. The base URL of the LM Studio server. Defaults to "http://localhost:1234".
`timeout`	Numeric. The number of seconds to wait for the request before giving up. Defaults to 2. A host that opens the port and never answers reports `FALSE` after this many seconds.
`token`	Character or `NULL`. An API token for a server that requires authentication. `NULL` reads the `rlmstudio.token` option and then the `RLMSTUDIO_API_TOKEN` environment variable. See rlmstudio_token.

Details

Use this in place of a port check. Another process holding the port, a server that has not finished starting, and a server that rejects the token all answer the port and all report `FALSE` here.

Value

TRUE or FALSE, always one value. This function raises no condition of its own for a network failure, a refused connection, an unparsable body, or a failed status. All of those report FALSE. A token that is not one character string and not NULL still aborts, because that is a fault in the call rather than a fact about the server.

Call faults that abort

A fault in the call is not a fact about the server, so it aborts rather than reporting FALSE. These messages come from the packages underneath. The httr2 ones name httr2's own arguments, url for host and seconds for timeout. The curl one names no argument at all. Six such faults are named below.

- A host of NULL. httr2 reports that url must be a single string, not NULL.
- A host of more than one string. httr2 reports that url must be a single string, not a character vector.
- A host that is a character NA. httr2 reports that url must be a single string, not a character NA.
- A host that is one string but cannot be parsed as a URL. curl reports that it failed to parse the URL and names the reason. The wording of the reason depends on the libcurl version. For a host holding a space, the reason is for example "Malformed input to a URL function" or "Bad hostname". For an empty host, it says that the URL has no host, for example "No host part in the URL" or "No host present".
- A timeout below one millisecond. httr2 reports that seconds must be greater than 1 ms.
- A timeout that is not one number, such as a string or a vector of two. httr2 reports that seconds must be a number, and names either the value or its type.

Those six are not the whole list. Any host that is not one string aborts the same way, whatever the reason, and httr2 names either the value or its type. A host of 1 gives "not the number 1". A host of list("a") gives "not a list". A host of character(0) gives "not an empty character vector".

The token fault named above aborts the same way. It comes from this package rather than from httr2.

See Also

lms_server_start() to start the server. lms_server_status() for what the CLI reports about it.

Examples

```
## Not run:
lms_server_start(port = 1234)

if (lms_server_ready()) {
  list_models()
}

# A server on another port, with a shorter wait
lms_server_ready(host = "http://localhost:8080", timeout = 0.5)

## End(Not run)
```

`lms_server_start` *Start the LM Studio local server*

Description

Launches the LM Studio local server via the CLI, allowing you to interact with loaded models via HTTP API calls.

Usage

```
lms_server_start(
  port = NULL,
  cors = FALSE,
  wait = 10,
  host = NULL,
  token = NULL
)
```

Arguments

`port`	Numeric or `NULL`. Port to run the server on. It must be one whole number from 1 to 65535, given as a number and not as an array or a string. `NULL`, the default, lets LM Studio use the last used port. The functions that send requests ask `http://localhost:1234` by default, so pass `port = 1234` unless you give those functions a `host`.
`cors`	Logical. Enable CORS support for web application development. Must be `TRUE` or `FALSE`. Defaults to `FALSE`. Any other value, `NULL` and `NA` included, aborts before the `lms` CLI runs.
`wait`	Numeric. How many seconds to keep asking the REST API whether it is ready. Defaults to 10. With `wait = 0` the function sends no readiness request and returns as soon as the CLI does. A `wait` that is not one number, zero or more, aborts before the CLI runs.
`host`	Character or `NULL`. The base URL to ask. This says where to look for the server that was started. It does not change where the CLI starts it, which only `port` does. `NULL` picks a host as described below.
`token`	Character or `NULL`. An API token for the readiness request, for a server that requires authentication. `NULL` reads the `rlmstudio.token` option and then the `RLMSTUDIO_API_TOKEN` environment variable. See rlmstudio_token.

Details

The CLI returns before the REST API answers. By default this function then keeps asking the REST API whether it is ready, for about `wait` seconds, and returns once it answers. A script that calls the REST API on the next line therefore no longer reports a missing server on a healthy machine. The budget is not a hard cap. No new request starts once `wait` seconds have passed, but a request already in flight is allowed one second to finish. With no `host` and no `port`, the function first asks the CLI which port the server uses, and that read runs before the `wait` seconds start to count.

Value

Invisibly returns an integer representing the system exit code (`0` for success).

Which host the wait asks

The host is picked in this order.

- A `host` you give wins.
- With no `host` and a `port`, the host is `http://localhost:<port>`.
- With neither, the port that `lms_server_status(json = TRUE)` reports is read, and the host is `http://localhost:` plus that port.

The request carries `token`, read as described under that argument.

Five faults in the call abort before the CLI runs, with any `wait`: a bad `wait`, `port`, `cors`, `host`, or `token`. A bad `host` is one that is not `NULL` and that the readiness request cannot be built from, such as a vector of two strings, `NA`, an empty string, or `"localhost:1234"`, which lacks `http://`. That message names `host` and quotes the reason httr2 or curl gave. A bad `token` is one that is not one character string and not `NULL`.

If the CLI refuses the start and exits with a status other than 0, the function aborts. The message gives the exit code and quotes what the CLI wrote, after "The CLI said:". The quoted text is the stderr text, or the stdout text if stderr holds only whitespace and escape codes. A byte that is not valid UTF-8 shows as <xx>, its hex value. ANSI escape codes, such as color codes, cursor codes, and terminal links, are removed, and each run of whitespace becomes one space. A text longer than 1000 characters keeps at most its last 1000 characters, after "...".

A wait that runs out does not abort. The server was already started and that cannot be undone, so the function raises a warning and returns the CLI exit code. A call with no `host` and no `port` whose port read yields nothing raises its own warning and sends no readiness request. If the readiness check itself aborts during the wait, the function raises a third warning. It names the host, quotes the abort message, and the function returns the CLI exit code. None of the three warnings is silenced by the `rlmstudio.quiet` option.

See Also

LM Studio CLI Server Start Documentation. `lms_server_ready()` for the readiness check this function calls.

Examples

```
## Not run:
# Start the server on port 1234, where the functions that send requests
# look by default, and wait for the REST API
lms_server_start(port = 1234)

# Start the server on another port with CORS enabled. The functions
# that send requests then need host = "http://localhost:8080".
lms_server_start(port = 8080, cors = TRUE)

# Return as soon as the CLI does, without waiting
lms_server_start(port = 1234, wait = 0)
```

```
# Wait longer, and ask a host the rules above would not pick
lms_server_start(port = 1234, wait = 60, host = "http://127.0.0.1:1234")

## End(Not run)
```

`lms_server_status` *Check the status of the LM Studio server*

Description

Displays the current status of the LM Studio local server via the CLI, including whether it is running and its configuration.

Usage

```
lms_server_status(
  json = FALSE,
  verbose = FALSE,
  quiet = FALSE,
  log_level = NULL
)
```

Arguments

`json`	`TRUE` or `FALSE`. Output the status in machine-readable JSON format. Any other value, `NULL` and `NA` included, aborts before the `lms` CLI runs.
`verbose`	`TRUE` or `FALSE`. Enable detailed logging output. Any other value, `NULL` and `NA` included, aborts before the `lms` CLI runs.
`quiet`	`TRUE` or `FALSE`. `TRUE` passes `--quiet` to the `lms` CLI, which suppresses all logging output. The `rlmstudio.quiet` option does not change it. Any other value, `NULL` and `NA` included, aborts before the `lms` CLI runs.
`log_level`	Character. The level of logging to use (e.g., "info", "debug").

Details

You can only use one logging control flag at a time (`verbose`, `quiet`, or `log_level`).

Value

By default, returns a character vector containing the raw CLI output. If `json = TRUE` and the `jsonlite` package is available, it returns a parsed list or `data.frame` of the status configuration.

See Also

LM Studio CLI Server Status Documentation

Examples

```
## Not run:
lms_server_start(port = 1234)

# Get basic status string
lms_server_status()

# Get status as a parsed JSON data frame
lms_server_status(json = TRUE)

## End(Not run)
```

`lms_server_stop` *Stop the LM Studio local server*

Description

Stops the currently running LM Studio local server via the CLI.

Usage

```
lms_server_stop()
```

Details

If the CLI exits with a status other than 0, the function reads what the CLI wrote. If the text says "not running", the function prints an info message and returns the exit code invisibly, with no abort. Letter case does not matter. With no server running, the CLI exits with status 1 and says so. `lms_daemon_stop()` with `force = TRUE` shows the same message.

Any other failure aborts. The message gives the exit code and quotes what the CLI wrote, after "The CLI said:". The quoted text is the stderr text, or the stdout text if stderr holds only whitespace and escape codes. A byte that is not valid UTF-8 shows as <xx>, its hex value. ANSI escape codes, such as color codes, cursor codes, and terminal links, are removed, and each run of whitespace becomes one space. A text longer than 1000 characters keeps at most its last 1000 characters, after "...".

Value

Invisibly returns an integer representing the system exit code (`0` for success).

See Also

LM Studio CLI Server Stop Documentation

Examples

```
## Not run:
lms_server_start(port = 1234)
lms_server_stop()

## End(Not run)
```

`lms_unload` *Unload a model from memory via REST API*

Description

Unload a model from memory via REST API

Usage

```
lms_unload(model, host = "http://localhost:1234", ..., token = NULL)
```

Arguments

`model`	Character. Unique identifier (`instance_id`) of the model instance to unload. Must be one name, given as a single string. The string must be valid in its declared encoding and not marked `"bytes"`. A class, names, and the S4 bit are removed before the name is sent.
`host`	Character. The host address of the local server. Defaults to "http://localhost:1234".
`...`	Additional arguments passed to the API request body.
`token`	Character or `NULL`. An API token for a server that requires authentication. `NULL` reads the `rlmstudio.token` option and then the `RLMSTUDIO_API_TOKEN` environment variable. See rlmstudio_token.

Value

Invisibly returns a character string representing the unloaded `instance_id` upon success. It is a plain string, with no class, names, or S4 bit.

Server not running

Functions that call the LM Studio REST API open a TCP connection to the hostname and port named in `host` before they send the request. A function that checks its own arguments does that first, so a bad `model`, `job_id`, `input`, `inputs`, `messages`, `schema`, `ttl`, `previous_response_id`, or `batch_size`, a bad type of `list_models()` or `list_instances()`, a bad `TRUE` or `FALSE` argument such as `simplify`, `logprobs`, `quiet`, `force`, or `store`, a `store` on the `"openai"` route of `lms_chat()` or `lms_chat_batch()`, a `stream` in the `...` of a chat function, a `logprobs` in the `...` of `lms_chat_batch()` or `lms_chat_native()`, a value in the `...` of `lms_chat_batch()` that `lms_chat()` reads as an argument already given, an `instructions` or `messages` in the `...` of `lms_chat()` or `lms_chat_batch()` on the route where `lms_chat()` sets it, or a name, id, or type

string that is not valid in its encoding or is marked as bytes, aborts with an argument message and no condition class even when the server is down. A condition of class `rlmstudio_no_server` is raised when that connection cannot be opened. A refused connection raises it. So do an address the package cannot parse and a hostname that does not resolve. An address that neither accepts nor refuses the connection also raises it. That case waits for the operating system to give up, which can take a minute. If the hostname of `host` is `localhost`, in any letter case, or `127.0.0.1`, start the server with `lms_server_start()` and the port of `host`. For the default `host`, that is `lms_server_start(port = 1234)`. With no `port`, LM Studio uses the port that it used last. For any other `host`, start the server on the machine that `host` names, or give `host` the address that your server listens on. A `host` that does not parse gets this second hint too. The condition message gives the same hint for the `host` of the call.

The check reads the port and nothing else. Any process holding that port accepts the connection, so the condition is not raised even though no LM Studio server is there. The call then does not raise `rlmstudio_no_server`, and what it does depends on what answers. On a status-200 body that does not parse as JSON, the chat functions, `lms_embed()`, `list_models()`, `list_instances()`, `lms_load()`, `lms_download()`, `lms_download_status()`, and `lms_unload_all()` raise `rlmstudio_bad_response`. `lms_unload()` does not read the body, so it can report success. A body that parses as JSON but has another shape can come back unchanged with `simplify = FALSE`. With `simplify = TRUE`, the chat functions and `lms_embed()` raise `rlmstudio_bad_response` for it. `list_models()` and `list_instances()` raise it for a model list with another shape, and so do `lms_unload_all()` and `lms_load()` without `force = TRUE`, which read that list. `lms_chat_openai()` and `lms_chat_openresponses()` raise it for such a model list too, with either setting of `simplify`, through the model lookup that a reply from another model starts. `lms_chat()` raises it through them. `lms_embed()` raises it in the same way for the inputs of that request. `lms_load()`, `lms_download()`, and `lms_download_status()` raise it for a reply of their own with another shape, such as `{}`. A process that does not answer in HTTP gives an `httr2_failure` error. Use `lms_server_ready()` for the stronger test: it asks the host for a model list and reports `TRUE` only for a model list that `list_models()` can read.

`lms_chat_batch()` checks the server once before its first input, and `lms_chat()` checks it again for each input. If that check finds the server gone during the batch, the batch aborts with `rlmstudio_no_server`, and no request goes out after that. The condition then carries a `results` field, a list as long as `inputs`. Its elements before the lost input hold the values that `format = "list"` returns for those inputs. The element of the lost input and every element after it are `NULL`. The check before the first input adds no `results` field. A connection that fails after the check passes, such as a server that stops during a request, raises an `httr2_failure` error instead. That error aborts the batch and carries no `results` field.

`lms_embed()` checks the server before each request, and a request carries at most `batch_size` inputs. If a check after the first request, or the check of a model-list lookup, finds the server gone, the call aborts with `rlmstudio_no_server`. Once a request has succeeded, the condition carries a `results` field. With `simplify = TRUE`, `results` is a matrix with `NA` in each row whose embedding did not arrive. With `simplify = FALSE`, it is a list with one element per batch, with `NULL` in the element of the request that ended the call and in every element after it.

API failure

A condition of class `rlmstudio_api_error` is raised when a REST call returns a response that the wrapper treats as a failure. The condition carries a `status` field, which holds the HTTP response status as an integer. It also carries a `code` field. The field holds the string at `error.code` of the response body, such as `"model_not_found"`. It is `NULL` when the body does not parse, when `error`

is not a JSON object, or when its code is not one string. A server_message field holds the error text as the server sent it: the error string of the body, or else the string at error.message. It is NULL when the body does not parse or holds neither string.

On LM Studio 0.4.26+4 with no model loaded, /v1/chat/completions, /v1/responses, and /v1/embeddings answered a model name that the server could not find with status 400, no code, and a server_message that starts with "No models loaded".

lms_chat_batch() aborts on it when its status is 401, 403, or 404, when its status is 400 and its code is "model_not_found", and when its status is 400 and its server_message starts with "No models loaded", in that letter case. No request goes out after that input. The condition then carries a results field that follows the rule for a lost server in the "Server not running" section: its elements before the failed input hold the values that format = "list" returns for those inputs, and the element of the failed input and every element after it are NULL. For any other status, the element of the failed input holds the condition, or NA where the result is text, and the batch warns once and goes on. See the details of lms_chat_batch().

lms_embed() follows the same rule for each request, except for the code "model_not_found". A 401, 403, or 404 aborts the call, and so does a 400 whose server_message starts with "No models loaded". Once a request has succeeded, the condition carries a results field, a matrix or a list as the "Server not running" section describes. Any other status, a 400 with the code "model_not_found" included, fails the inputs of that request alone, and the call warns once and goes on. If every request fails, the call aborts with the first condition. See the details of lms_embed().

Note

If you have loaded multiple instances of the same model using force = TRUE in lms_load(), the server assigns them unique instance identifiers (e.g., "google/gemma-3-1b" and "google/gemma-3-1b:2"). Passing the base model name to lms_unload() will only unload the primary instance. To unload duplicate instances, you must provide their exact instance_id, or use lms_unload_all() to clear everything.

See Also

LM Studio Unload Model API

Examples

```
## Not run:
lms_server_start(port = 1234)
lms_download("google/gemma-3-1b")
lms_load("google/gemma-3-1b")

# Unload a single specific model
lms_unload("google/gemma-3-1b")

## End(Not run)
```

lms_unload_all *Unload all models from memory*

Description

Retrieves a list of all currently loaded models and unloads them one by one.

Usage

```
lms_unload_all(host = "http://localhost:1234", ..., token = NULL)
```

Arguments

host	Character. The host address of the local server. Defaults to "http://localhost:1234".
...	Additional arguments passed to the API request body for each unload request.
token	Character or NULL. An API token for a server that requires authentication. NULL reads the rlmstudio.token option and then the RLMSTUDIO_API_TOKEN environment variable. See rlmstudio_token.

Details

This function calls list_models() to find the loaded instances, then calls lms_unload() once for each one. It raises rlmstudio_no_server itself, before the first call. It can raise rlmstudio_api_error through list_models() and through lms_unload(). It can raise rlmstudio_bad_response through list_models().

Value

Invisibly returns a character vector of the instance_ids that were successfully unloaded. If no models were currently loaded, it invisibly returns NULL.

Server not running

Functions that call the LM Studio REST API open a TCP connection to the hostname and port named in host before they send the request. A function that checks its own arguments does that first, so a bad model, job_id, input, inputs, messages, schema, ttl, previous_response_id, or batch_size, a bad type of list_models() or list_instances(), a bad TRUE or FALSE argument such as simplify, logprobs, quiet, force, or store, a store on the "openai" route of lms_chat() or lms_chat_batch(), a stream in the ... of a chat function, a logprobs in the ... of lms_chat_batch() or lms_chat_native(), a value in the ... of lms_chat_batch() that lms_chat() reads as an argument already given, an instructions or messages in the ... of lms_chat() or lms_chat_batch() on the route where lms_chat() sets it, or a name, id, or type string that is not valid in its encoding or is marked as bytes, aborts with an argument message and no condition class even when the server is down. A condition of class rlmstudio_no_server is raised when that connection cannot be opened. A refused connection raises it. So do an address the package cannot parse and a hostname that does not resolve. An address that neither accepts nor refuses the connection also raises it. That case waits for the operating system to give up, which can take a

minute. If the hostname of host is localhost, in any letter case, or 127.0.0.1, start the server with lms_server_start() and the port of host. For the default host, that is lms_server_start(port = 1234). With no port, LM Studio uses the port that it used last. For any other host, start the server on the machine that host names, or give host the address that your server listens on. A host that does not parse gets this second hint too. The condition message gives the same hint for the host of the call.

The check reads the port and nothing else. Any process holding that port accepts the connection, so the condition is not raised even though no LM Studio server is there. The call then does not raise rlmstudio_no_server, and what it does depends on what answers. On a status-200 body that does not parse as JSON, the chat functions, lms_embed(), list_models(), list_instances(), lms_load(), lms_download(), lms_download_status(), and lms_unload_all() raise rlmstudio_bad_response. lms_unload() does not read the body, so it can report success. A body that parses as JSON but has another shape can come back unchanged with simplify = FALSE. With simplify = TRUE, the chat functions and lms_embed() raise rlmstudio_bad_response for it. list_models() and list_instances() raise it for a model list with another shape, and so do lms_unload_all() and lms_load() without force = TRUE, which read that list. lms_chat_openai() and lms_chat_openresponses() raise it for such a model list too, with either setting of simplify, through the model lookup that a reply from another model starts. lms_chat() raises it through them. lms_embed() raises it in the same way for the inputs of that request. lms_load(), lms_download(), and lms_download_status() raise it for a reply of their own with another shape, such as {}. A process that does not answer in HTTP gives an httr2_failure error. Use lms_server_ready() for the stronger test: it asks the host for a model list and reports TRUE only for a model list that list_models() can read.

lms_chat_batch() checks the server once before its first input, and lms_chat() checks it again for each input. If that check finds the server gone during the batch, the batch aborts with rlmstudio_no_server, and no request goes out after that. The condition then carries a results field, a list as long as inputs. Its elements before the lost input hold the values that format = "list" returns for those inputs. The element of the lost input and every element after it are NULL. The check before the first input adds no results field. A connection that fails after the check passes, such as a server that stops during a request, raises an httr2_failure error instead. That error aborts the batch and carries no results field.

lms_embed() checks the server before each request, and a request carries at most batch_size inputs. If a check after the first request, or the check of a model-list lookup, finds the server gone, the call aborts with rlmstudio_no_server. Once a request has succeeded, the condition carries a results field. With simplify = TRUE, results is a matrix with NA in each row whose embedding did not arrive. With simplify = FALSE, it is a list with one element per batch, with NULL in the element of the request that ended the call and in every element after it.

API failure

A condition of class rlmstudio_api_error is raised when a REST call returns a response that the wrapper treats as a failure. The condition carries a status field, which holds the HTTP response status as an integer. It also carries a code field. The field holds the string at error.code of the response body, such as "model_not_found". It is NULL when the body does not parse, when error is not a JSON object, or when its code is not one string. A server_message field holds the error text as the server sent it: the error string of the body, or else the string at error.message. It is NULL when the body does not parse or holds neither string.

On LM Studio 0.4.26+4 with no model loaded, /v1/chat/completions, /v1/responses, and

/v1/embeddings answered a model name that the server could not find with status 400, no code, and a server_message that starts with "No models loaded".

lms_chat_batch() aborts on it when its status is 401, 403, or 404, when its status is 400 and its code is "model_not_found", and when its status is 400 and its server_message starts with "No models loaded", in that letter case. No request goes out after that input. The condition then carries a results field that follows the rule for a lost server in the "Server not running" section: its elements before the failed input hold the values that format = "list" returns for those inputs, and the element of the failed input and every element after it are NULL. For any other status, the element of the failed input holds the condition, or NA where the result is text, and the batch warns once and goes on. See the details of lms_chat_batch().

lms_embed() follows the same rule for each request, except for the code "model_not_found". A 401, 403, or 404 aborts the call, and so does a 400 whose server_message starts with "No models loaded". Once a request has succeeded, the condition carries a results field, a matrix or a list as the "Server not running" section describes. Any other status, a 400 with the code "model_not_found" included, fails the inputs of that request alone, and the call warns once and goes on. If every request fails, the call aborts with the first condition. See the details of lms_embed().

Malformed response

A condition of class rlmstudio_bad_response is raised when the server answers with a status the wrapper accepts and a body the wrapper cannot read. It is raised where a wrapper checks the body before it reshapes it, rather than indexing straight into whatever arrived. Eleven functions raise it for a body they cannot read: lms_embed(), lms_chat(), lms_chat_native(), lms_chat_openresponses(), lms_chat_openai(), list_models(), list_instances(), lms_load(), lms_download(), lms_download_status(), and lms_unload_all(). lms_chat() raises it through the chat function it calls. lms_unload_all() raises it through list_models(), and so does lms_load() unless force = TRUE. lms_chat_batch() raises it only as rlmstudio_model_mismatch, which the "Reply from another model" section of lms_chat_openai() describes.

All eleven raise it for a status-200 body that does not parse as JSON, such as an HTML page from a proxy, JSON text that stops part way, or an empty body. In the functions that take simplify, the body is parsed before simplify is read, so the condition is raised whatever simplify is. The body is parsed by its content and not by its Content-Type header, so valid JSON under text/plain is read as JSON. The body is read as JSON text and nothing else. A body whose text is a URL or the path of a file does not parse, and the package does not fetch the URL or read the file. The message says that the body did not parse as JSON and that something other than LM Studio may be answering on the host. It does not hold the body text.

The condition carries a status field, which holds the HTTP response status as an integer. Today the status is always 200: each of these functions reads the body only after a 200, and reports every other status as an rlmstudio_api_error instead.

Malformed model list

list_models() also raises it for a status-200 model list with the wrong shape. lms_unload_all() and lms_load() without force = TRUE raise it through list_models(). lms_chat_openai(), lms_chat_openresponses(), and lms_embed() apply these rules to the model list of their model lookup, as the "Reply from another model" section of lms_chat_openai() describes. A model list must follow four rules. Each field is read by its exact name, so a field named keyX does not stand in for key.

1. The body is a JSON object whose `models` field is an array. The array can be empty.
2. Each entry of `models` is a JSON object. Its `type` and `key` are strings, and its `loaded_instances` is an array.
3. The `size_bytes` of an entry is a number, or absent, or `null`. So is its `max_context_length`.
4. Each entry of `loaded_instances` is a JSON object whose `id` is a string with a character that is not whitespace.

The rules are checked before the `type` and `loaded` filters, so an entry that the filters drop can still raise the condition. The message names the field or entry that broke a rule. `lms_server_ready()` applies the same rules and returns `FALSE` for a body that breaks one.

`list_instances()` applies the four rules too, and three more. They cover only an entry whose `type` is in its `type` argument and that has at least one loaded instance. The `display_name` of such an entry is a string, or absent, or `null`. The `remaining_ttl_seconds` of each of its instances is a number, or absent, or `null`. The `config` of each of its instances is a JSON object, or absent, or `null`. `list_models()` and `lms_server_ready()` do not apply these three rules.

See Also

`lms_unload`

Examples

```
## Not run:
lms_server_start(port = 1234)
lms_download("google/gemma-3-1b")
lms_load("google/gemma-3-1b")

# Unload all currently loaded models to clear VRAM
lms_unload_all()

## End(Not run)
```

`rlmstudio-conditions` *Conditions raised by rlmstudio*

Description

The functions in this package that talk to the LM Studio REST API raise four condition classes of their own. One of them, `rlmstudio_model_mismatch`, is also an `rlmstudio_bad_response`, and the "Reply from another model" section describes it. Each one is raised through `cli::cli_abort()`, so each one is an R error that you can catch by class with `base::tryCatch()`. The chat functions on the OpenAI route also give two warning classes of their own. The "Cut-off reply" section describes `rlmstudio_reply_cut_off`, and `lms_chat_openai()` describes `rlmstudio_no_logprobs`.

Server not running

Functions that call the LM Studio REST API open a TCP connection to the hostname and port named in `host` before they send the request. A function that checks its own arguments does that first, so a bad `model`, `job_id`, `input`, `inputs`, `messages`, `schema`, `ttl`, `previous_response_id`, or `batch_size`, a bad type of `list_models()` or `list_instances()`, a bad `TRUE` or `FALSE` argument such as `simplify`, `logprobs`, `quiet`, `force`, or `store`, a `store` on the `"openai"` route of `lms_chat()` or `lms_chat_batch()`, a `stream` in the `...` of a chat function, a `logprobs` in the `...` of `lms_chat_batch()` or `lms_chat_native()`, a value in the `...` of `lms_chat_batch()` that `lms_chat()` reads as an argument already given, an `instructions` or `messages` in the `...` of `lms_chat()` or `lms_chat_batch()` on the route where `lms_chat()` sets it, or a name, id, or type string that is not valid in its encoding or is marked as bytes, aborts with an argument message and no condition class even when the server is down. A condition of class `rlmstudio_no_server` is raised when that connection cannot be opened. A refused connection raises it. So do an address the package cannot parse and a hostname that does not resolve. An address that neither accepts nor refuses the connection also raises it. That case waits for the operating system to give up, which can take a minute. If the hostname of `host` is `localhost`, in any letter case, or `127.0.0.1`, start the server with `lms_server_start()` and the port of `host`. For the default `host`, that is `lms_server_start(port = 1234)`. With no `port`, LM Studio uses the port that it used last. For any other `host`, start the server on the machine that `host` names, or give `host` the address that your server listens on. A `host` that does not parse gets this second hint too. The condition message gives the same hint for the `host` of the call.

The check reads the port and nothing else. Any process holding that port accepts the connection, so the condition is not raised even though no LM Studio server is there. The call then does not raise `rlmstudio_no_server`, and what it does depends on what answers. On a status-200 body that does not parse as JSON, the chat functions, `lms_embed()`, `list_models()`, `list_instances()`, `lms_load()`, `lms_download()`, `lms_download_status()`, and `lms_unload_all()` raise `rlmstudio_bad_response`. `lms_unload()` does not read the body, so it can report success. A body that parses as JSON but has another shape can come back unchanged with `simplify = FALSE`. With `simplify = TRUE`, the chat functions and `lms_embed()` raise `rlmstudio_bad_response` for it. `list_models()` and `list_instances()` raise it for a model list with another shape, and so do `lms_unload_all()` and `lms_load()` without `force = TRUE`, which read that list. `lms_chat_openai()` and `lms_chat_openresponses()` raise it for such a model list too, with either setting of `simplify`, through the model lookup that a reply from another model starts. `lms_chat()` raises it through them. `lms_embed()` raises it in the same way for the inputs of that request. `lms_load()`, `lms_download()`, and `lms_download_status()` raise it for a reply of their own with another shape, such as `{}`. A process that does not answer in HTTP gives an `httr2_failure` error. Use `lms_server_ready()` for the stronger test: it asks the host for a model list and reports `TRUE` only for a model list that `list_models()` can read.

`lms_chat_batch()` checks the server once before its first input, and `lms_chat()` checks it again for each input. If that check finds the server gone during the batch, the batch aborts with `rlmstudio_no_server`, and no request goes out after that. The condition then carries a `results` field, a list as long as `inputs`. Its elements before the lost input hold the values that `format = "list"` returns for those inputs. The element of the lost input and every element after it are `NULL`. The check before the first input adds no `results` field. A connection that fails after the check passes, such as a server that stops during a request, raises an `httr2_failure` error instead. That error aborts the batch and carries no `results` field.

`lms_embed()` checks the server before each request, and a request carries at most `batch_size` inputs. If a check after the first request, or the check of a model-list lookup, finds the server gone,

the call aborts with rlmstudio_no_server. Once a request has succeeded, the condition carries a results field. With simplify = TRUE, results is a matrix with NA in each row whose embedding did not arrive. With simplify = FALSE, it is a list with one element per batch, with NULL in the element of the request that ended the call and in every element after it.

API failure

A condition of class rlmstudio_api_error is raised when a REST call returns a response that the wrapper treats as a failure. The condition carries a status field, which holds the HTTP response status as an integer. It also carries a code field. The field holds the string at error.code of the response body, such as "model_not_found". It is NULL when the body does not parse, when error is not a JSON object, or when its code is not one string. A server_message field holds the error text as the server sent it: the error string of the body, or else the string at error.message. It is NULL when the body does not parse or holds neither string.

On LM Studio 0.4.26+4 with no model loaded, /v1/chat/completions, /v1/responses, and /v1/embeddings answered a model name that the server could not find with status 400, no code, and a server_message that starts with "No models loaded".

lms_chat_batch() aborts on it when its status is 401, 403, or 404, when its status is 400 and its code is "model_not_found", and when its status is 400 and its server_message starts with "No models loaded", in that letter case. No request goes out after that input. The condition then carries a results field that follows the rule for a lost server in the "Server not running" section: its elements before the failed input hold the values that format = "list" returns for those inputs, and the element of the failed input and every element after it are NULL. For any other status, the element of the failed input holds the condition, or NA where the result is text, and the batch warns once and goes on. See the details of lms_chat_batch().

lms_embed() follows the same rule for each request, except for the code "model_not_found". A 401, 403, or 404 aborts the call, and so does a 400 whose server_message starts with "No models loaded". Once a request has succeeded, the condition carries a results field, a matrix or a list as the "Server not running" section describes. Any other status, a 400 with the code "model_not_found" included, fails the inputs of that request alone, and the call warns once and goes on. If every request fails, the call aborts with the first condition. See the details of lms_embed().

Malformed response

A condition of class rlmstudio_bad_response is raised when the server answers with a status the wrapper accepts and a body the wrapper cannot read. It is raised where a wrapper checks the body before it reshapes it, rather than indexing straight into whatever arrived. Eleven functions raise it for a body they cannot read: lms_embed(), lms_chat(), lms_chat_native(), lms_chat_openresponses(), lms_chat_openai(), list_models(), list_instances(), lms_load(), lms_download(), lms_download_status(), and lms_unload_all(). lms_chat() raises it through the chat function it calls. lms_unload_all() raises it through list_models(), and so does lms_load() unless force = TRUE. lms_chat_batch() raises it only as rlmstudio_model_mismatch, which the "Reply from another model" section of lms_chat_openai() describes.

All eleven raise it for a status-200 body that does not parse as JSON, such as an HTML page from a proxy, JSON text that stops part way, or an empty body. In the functions that take simplify, the body is parsed before simplify is read, so the condition is raised whatever simplify is. The body is parsed by its content and not by its Content-Type header, so valid JSON under text/plain is read as JSON. The body is read as JSON text and nothing else. A body whose text is a URL

or the path of a file does not parse, and the package does not fetch the URL or read the file. The message says that the body did not parse as JSON and that something other than LM Studio may be answering on the host. It does not hold the body text.

The condition carries a `status` field, which holds the HTTP response status as an integer. Today the status is always 200: each of these functions reads the body only after a 200, and reports every other status as an `rlmstudio_api_error` instead.

Malformed model list

`list_models()` also raises it for a status-200 model list with the wrong shape. `lms_unload_all()` and `lms_load()` without `force = TRUE` raise it through `list_models()`. `lms_chat_openai()`, `lms_chat_openresponses()`, and `lms_embed()` apply these rules to the model list of their model lookup, as the "Reply from another model" section of `lms_chat_openai()` describes. A model list must follow four rules. Each field is read by its exact name, so a field named `keyX` does not stand in for `key`.

1. The body is a JSON object whose `models` field is an array. The array can be empty.
2. Each entry of `models` is a JSON object. Its `type` and `key` are strings, and its `loaded_instances` is an array.
3. The `size_bytes` of an entry is a number, or absent, or `null`. So is its `max_context_length`.
4. Each entry of `loaded_instances` is a JSON object whose `id` is a string with a character that is not whitespace.

The rules are checked before the `type` and `loaded` filters, so an entry that the filters drop can still raise the condition. The message names the field or entry that broke a rule. `lms_server_ready()` applies the same rules and returns `FALSE` for a body that breaks one.

`list_instances()` applies the four rules too, and three more. They cover only an entry whose `type` is in its `type` argument and that has at least one loaded instance. The `display_name` of such an entry is a string, or absent, or `null`. The `remaining_ttl_seconds` of each of its instances is a number, or absent, or `null`. The `config` of each of its instances is a JSON object, or absent, or `null`. `list_models()` and `lms_server_ready()` do not apply these three rules.

Malformed load or download reply

`lms_load()`, `lms_download()`, and `lms_download_status()` also raise it for a status-200 reply of their own with the wrong shape. Each reply must follow the rule of its function. Each field is read by its exact name. The rules check the type of a field and not its value, with four exceptions. The `status` of a load reply must be `"loaded"`. A download reply whose `status` is `"already_downloaded"` needs no `job_id`. A download reply whose `status` is `"failed"` always aborts. The `job_id` of any other download reply must hold a character that is not whitespace.

1. A reply of `lms_load()` is a JSON object whose `status` is the string `"loaded"`. With `echo_load_config = TRUE`, its `load_config` is also a JSON object.
2. A reply of `lms_download()` is a JSON object whose `status` is a string other than `"failed"`. If the status is not `"already_downloaded"`, its `job_id` is a string with a character that is not whitespace. For a `"failed"` status, the message says that LM Studio reports that the download failed. It names the reply's `job_id` if that is a string with a character that is not whitespace.

3. A reply of lms_download_status() is a JSON object whose job_id and status are strings. Its total_size_bytes, downloaded_bytes, and bytes_per_second are each a number, or absent, or null.

For the other faults, the message names the field that broke the rule, or it says that the body is not a JSON object. It also says that something other than LM Studio may be answering on the host.

Malformed embeddings

lms_embed() raises it on an embeddings block it cannot trust. The vectors it returns are placed by the index that the response reports, so a block with a missing, repeated, or out-of-range index would otherwise pair a vector with the wrong text and give back a matrix that is silently wrong.

lms_embed() reads each request on its own. A bad body, of either kind that the "Malformed response" section and this section describe, fails the inputs of that request alone, and the call warns once and goes on. Unless a stop that the details of lms_embed() list ends the call first, the call aborts with the condition only if every request fails, and then with the condition of the first. With simplify = TRUE, it also aborts with rlmstudio_bad_response for a request whose embeddings have another number of dimensions than those of an earlier request. That condition carries a results field, a matrix as the "Server not running" section describes. An rlmstudio_model_mismatch also ends the call, as the "Reply from another model" section says. See the details of lms_embed().

The messages of lms_embed() name simplify = FALSE, which returns the body unchanged, with three exceptions. A body that did not parse as JSON is checked before that argument is read, so its message points at the host instead. The messages of rlmstudio_model_mismatch and of a failed model-list lookup do not name it, because the check runs with either setting of simplify.

Malformed chat reply

lms_chat_native() and lms_chat_openresponses() raise it with simplify = TRUE when the reply holds no readable answer text. Both read the answer from the items of type "message" in the output array. They raise it when output is missing, empty, or not an array, or when an item in it is not a JSON object. They also raise it when no item has the type "message", as in a reply that holds only reasoning or a tool call. The text of a message must be one string. For lms_chat_native() that is the content of the item. For lms_chat_openresponses() it is the text of each part of type "output_text". For lms_chat_openresponses() only, the content of each message must be an array of JSON objects, and the messages together must hold at least one "output_text" part.

Apart from a body that does not parse as JSON, lms_chat_openai() raises it in four cases, all only with simplify = TRUE. The first case is a response whose choices field is missing, empty, or not an array, or whose first element is not a JSON object with a message object in it, so there is no reply to read. This case is raised with or without a schema, and with logprobs = TRUE as well. The second case is a reply that does not parse. A schema was given, logprobs = FALSE, and the reply content is not one string of valid JSON. The third case is reply content that is not one string, such as null, a missing content field, a number, or an array. A reply that holds only a tool call has null content. This case is raised without a schema, and with logprobs = TRUE with or without one. The fourth case is a cut-off reply. A schema was given, logprobs = FALSE, and the server reports the finish reason "length". This case is raised also when the reply content parses, because a reply that stops part way can parse to a wrong value, such as the first digit of a longer number. In the second, third, and fourth cases, if the server reports the finish reason "length", a length limit ended the reply. The limit is max_tokens or the context length of the model. The message then says so and names both.

With simplify = TRUE, lms_chat_native(), lms_chat_openresponses(), and lms_chat_openai() also raise it for a body that is a bare JSON value, such as 5, "s", or true. The message says that the response body is not a JSON object. A body of null gets the message about its missing output or choices field instead. lms_chat() can raise the condition through all three chat functions.

lms_chat_batch() does not abort on it, except on the subclass rlmstudio_model_mismatch, as the "Reply from another model" section of lms_chat_openai() says. The element of the failed input holds the condition, or NA where the result is text, and the batch warns once and goes on. See the details of lms_chat_batch().

A condition from lms_chat_openai() about its reply also carries two more fields. A condition from the model-list lookup does not, as the "Reply from another model" section of lms_chat_openai() says. The content field holds the reply content of the first choice, and the finish_reason field holds the finish reason of the first choice. Both are NULL for a response with no choices. In the third case, content holds the value that was read, which is NULL for null or missing content. For the second, third, and fourth cases, the message names the content field, so you can read what the model wrote without a second request. The other messages of the chat functions about the reply name simplify = FALSE, which returns the body unchanged, with one exception. A body that did not parse as JSON is checked before that argument is read, so its message points at the host instead. For such a body, the content and finish_reason fields of a condition from lms_chat_openai() are NULL. The messages of rlmstudio_model_mismatch and of the model-list lookup do not name simplify = FALSE, because the check runs with either setting of simplify.

Malformed logprobs

With simplify = TRUE and logprobs = TRUE, lms_chat_openresponses() also raises it for a logprobs value that breaks one of these rules. The logprobs value of each "output_text" part is checked. A logprobs value, token, logprob, or top_logprobs that is null or absent passes its rule. A null step or candidate breaks rule 2 or rule 6.

1. The value is an array.
2. Each step in the array is a JSON object.
3. The token of a step is a string.
4. The logprob of a step is a number.
5. The top_logprobs of a step is an array.
6. Each candidate in top_logprobs is a JSON object.
7. The token and logprob of each candidate follow rules 3 and 4.

The parts are checked in order, then the steps of a part, then the candidates of a step, one at a time. Within a step, rules 3, 4, and 5 come before the candidates. The message names the first broken rule that this order reaches. These checks run only after the text of every "output_text" part is read, so a reply that also has a bad text in any part gets the text message. Parts of other types, such as a refusal, are not checked, and with logprobs = FALSE no part is checked. Fields are read by their exact names, so a field whose name only starts with the one asked for, such as tokenX, reads as absent and gives NA in the data frame.

Cut-off reply

A warning of class rlmstudio_reply_cut_off is given when a length limit ended a reply that lms_chat_openai() returns as text. The server reports this with the finish reason "length". The

limit is `max_tokens` or the context length of the model, and the reply does not say which one. The message names both. The warning is given with `simplify = TRUE` for reply content that is one string, in three settings: no `schema`, `logprobs = TRUE`, and a `schema` with `logprobs = TRUE`. The call returns what the same reply returns without the cut-off. A `schema` reply with `logprobs = FALSE` raises `rlmstudio_bad_response` instead, as the "Malformed chat reply" section says. The warning and that abort read the finish reason of the first choice, which is the choice that the call reads. No other element of `choices` is read. With `simplify = FALSE`, the call returns the body with every choice and no warning.

`lms_chat()` gives the warning through `lms_chat_openai()` with `api_type = "openai"`. `lms_chat_batch()` gives one warning of this class for the whole batch in place of one for each input. It names the count and the positions of the cut-off inputs, and each of those elements keeps its reply. It comes after the warning about failed inputs. A batch that aborts with a `results` field gives this warning before the abort. It names the cut-off inputs whose replies the `results` field of the abort holds. The native and OpenResponses routes give no such warning.

The warning shows whatever `quiet` and the `rlmstudio.quiet` option say, because it is the only sign that an answer is not complete.

Reply from another model

`lms_chat_openai()`, `lms_chat_openresponses()`, and `lms_embed()` read the `model` field of each status-200 reply, with either setting of `simplify`. On LM Studio 0.4.25+1 with one chat model loaded, both chat routes answered a model name that the server could not find with status 200 and a reply from the loaded model. With two chat models loaded, they answered with status 400 and the code `"model_not_found"`, as the "API failure" section describes. On LM Studio 0.4.26+4 with one embedding model loaded, `/v1/embeddings` answered a name that it could not find with status 200 and vectors from the loaded embedding model. The `model` field names the loaded instance that answered. Its id can differ from the key that the call asked for.

If the `model` field equals the name that the call sent, the reply is accepted. If it differs, the call sends one request for the model list to `api/v1/models` on the same host, with the token of the call, and prints no message. The reply is accepted if a model whose key equals the asked name in any letter case has a loaded instance whose id is the `model` field. Otherwise the call aborts with a condition of class `rlmstudio_model_mismatch`. That condition is also an `rlmstudio_bad_response`, and its `status` is 200. Its `model` field holds the asked name, and its `reply_model` field holds the `model` field of the reply. The message names both. A condition from `lms_chat_openai()` also carries the `content` and `finish_reason` fields, both `NULL`.

Within one call of `lms_chat_batch()` or `lms_embed()`, a reply whose `model` field a lookup of that call already accepted for the asked name and the same host is accepted with no other request. Each such call, and each call of `lms_chat_openai()` or `lms_chat_openresponses()` outside a batch, starts with no accepted ids.

The check runs before the reply is read, so it also runs with `logprobs = TRUE`, with a `schema` on `lms_chat_openai()`, and for a reply with no answer text. In `lms_embed()`, it runs before the vectors are read. A reply is not checked if its body is not a JSON object, if it has no `model` field, or if its `model` is not one string or holds only whitespace. A body that is not a JSON object is returned with `simplify = FALSE`, and with `simplify = TRUE` it raises the error that the "Malformed chat reply" section describes, or for `lms_embed()` the "Malformed embeddings" section. If the instance that answered is unloaded before the model-list request, the call aborts, also when that instance belongs to the asked model.

If the model-list request fails, the check raises the condition of that failure. The message opens with the label of the function, followed by "because the model-list lookup failed". A status other than 200 raises `rlmstudio_api_error`. A body that does not parse as JSON, or that breaks a rule of a model list in the "Malformed model list" section, raises `rlmstudio_bad_response`. A server that the port check before the request cannot reach raises `rlmstudio_no_server`. A condition from the lookup does not carry the `content` and `finish_reason` fields, also when `lms_chat_openai()` raises it.

`lms_chat()` runs the check through `lms_chat_openai()` and `lms_chat_openresponses()`. `lms_chat_native()` does not check the reply. On LM Studio 0.4.25+1, `/api/v1/chat` answered a name that it could not find with status 404, which raises `rlmstudio_api_error`.

`lms_chat_batch()` aborts at the first input that raises `rlmstudio_model_mismatch`, on the `"openai"` and `"openresponses"` routes. It also aborts at an `rlmstudio_api_error` with status 400 whose `code` is `"model_not_found"`, and at one with status 400 whose `server_message` starts with "No models loaded". No request goes out after that input. In each case, the condition carries a `results` field that follows the rule for a lost server in the "Server not running" section.

`lms_embed()` aborts at the first request that raises `rlmstudio_model_mismatch`, and no request goes out after it. A failed model-list lookup that raises `rlmstudio_no_server`, or an `rlmstudio_api_error` that would end the call as an embeddings failure, such as one with status 401, 403, or 404, ends the call in the same way. Once a request has succeeded, the condition carries a `results` field, a matrix or a list as the "Server not running" section describes. Any other `rlmstudio_api_error` or `rlmstudio_bad_response` from the lookup fails the inputs of that request alone, as the "API failure" section describes for a failed request. An error of any other class, such as an `httr2_failure` from a server that stops during the lookup, aborts the call unchanged.

Examples

```
## Not run:
tryCatch(
  list_models(host = "http://localhost:9999"),
  rlmstudio_no_server = function(cnd) {
    message(conditionMessage(cnd))
  },
  rlmstudio_api_error = function(cnd) {
    message("The API call failed with status ", cnd$status)
  },
  rlmstudio_bad_response = function(cnd) {
    message("The response body could not be read: ", conditionMessage(cnd))
  }
)

## End(Not run)
```

`rlmstudio_token` *Authenticating to an LM Studio server*

Description

An LM Studio server can require an API token. Every function in this package that calls the LM Studio REST API takes a `token` argument and sends the value as a bearer token in the `Authorization` header. Create and manage the token in the LM Studio app.

Where the token comes from

A function reads three sources and uses the first one that holds a value.

1. The `token` argument of the function you call.
2. The `rlmstudio.token` option, set with `base::options()`.
3. The `RLMSTUDIO_API_TOKEN` environment variable.

When none of the three holds a value, the request carries no `Authorization` header. A source that is `NULL` or an empty string counts as unset. A `token` argument of any other shape, such as a vector of two strings or a number, aborts rather than falling through to the next source.

Keeping the token out of your output

The header is set with `httr2::req_auth_bearer_token()`, which marks it as redacted. Printing a request shows `<REDACTED>` in place of the value, and this package never puts the value into the message of a failed call.

Two things this package does not control can still show the value. The message of a failed call repeats the text the server sent, so a server that echoes the token back puts it there. An R error also prints a backtrace, which repeats your own calling line, so a token you write as a literal appears in it. Read the token from an environment variable to keep it out of both.

When the server rejects the call

A response with HTTP status 401 or 403 aborts with a condition of class `rlmstudio_api_error`, described in rlmstudio-conditions. If the request carried no token, the message tells you to set `RLMSTUDIO_API_TOKEN` or to pass `token`. If the request carried a token, the message tells you that the server rejected it.

See Also

rlmstudio-conditions

Examples

```
## Not run:
# Per call.
list_models(token = "your-token")

# For the session.
options(rlmstudio.token = "your-token")
list_models()

# For the machine, set RLMSTUDIO_API_TOKEN in your .Renviron file.
```

```
## End(Not run)
```

`with_lms_daemon` *Run code with the LM Studio daemon active*

Description

Temporarily starts the LM Studio headless daemon, executes the provided R expression, and then gracefully shuts the daemon and any active servers down. This is ideal for automated scripts and pipelines.

Usage

```
with_lms_daemon(code)
```

Arguments

`code`	An R expression to execute while the daemon is running.

Value

The result of the evaluated code.

Desktop Users

Be cautious using this wrapper if you already have the LM Studio GUI open. While the setup phase (`lms_daemon_start`) will succeed, the teardown phase (`lms_daemon_stop`) does not stop the daemon, because the CLI prevents programmatic shutdowns of the graphical interface. The teardown prints an info message, and the daemon keeps running. This wrapper is best reserved for strictly headless environments or fully automated scripts.

Examples

```
## Not run:
result <- with_lms_daemon({
  lms_load("llama-3.1-8b")
  lms_chat("llama-3.1-8b", input = "Hello world!")
})

## End(Not run)
```

Index

base::options(), *99*
base::tryCatch(), *91*

check_lms_version, 2
cli::cli_abort(), *91*

has_lms, 3
httr2::req_auth_bearer_token(), *99*

I(), *41, 42*
install_lmstudio, 3

jsonlite::parse_json(), *6*

list_instances, 4
list_instances(), *6–13*, *16–19*, *27–30*, *35–37*, *43–45*, *51–53*, *59–61*, *63, 64*, *68*, *70*, *74–76*, *85, 86*, *88–94*
list_models, 9
list_models(), *4*, *6–13*, *16–19*, *27–30*, *35–37*, *43–45*, *51–53*, *59–61*, *63, 64*, *68–70*, *74–76*, *79*, *85, 86*, *88–94*
lms_chat, 13
lms_chat(), *6–8*, *10–12*, *16–19*, *21–29*, *31–38*, *43, 44*, *46, 47*, *50–52*, *54, 55*, *59–61*, *63, 64*, *68–71*, *74, 75*, *85, 86*, *88–90*, *92, 93*, *96–98*
lms_chat_batch, 23
lms_chat_batch(), *6–8*, *10–12*, *14–19*, *21, 22*, *27–29*, *31–38*, *43, 44*, *46, 47*, *49–52*, *54, 55*, *59–61*, *63, 64*, *68–71*, *73–75*, *85–90*, *92, 93*, *96–98*
lms_chat_native, 33
lms_chat_native(), *6*, *8*, *10*, *12*, *15, 16*, *18, 19*, *22*, *27*, *29, 30*, *33*, *35*, *37, 38*, *43–47*, *49–55*, *59*, *61*, *63, 64*, *68*, *70, 71*, *74, 75*, *85*, *88*, *90*, *92, 93*, *95, 96*, *98*
lms_chat_openai, 38
lms_chat_openai(), *7, 8*, *11, 12*, *14–22*, *28–34*, *36–38*, *43–47*, *50–55*, *60, 61*, *63, 64*, *68*, *70, 71*, *74–76*, *86*, *89–98*
lms_chat_openresponses, 48
lms_chat_openresponses(), *7, 8*, *11, 12*, *15–22*, *28–33*, *35–38*, *42–47*, *51–55*, *60, 61*, *63, 64*, *68*, *70, 71*, *74–76*, *86*, *89, 90*, *92–98*
lms_daemon_start, 56
lms_daemon_status, 57
lms_daemon_stop, 57
lms_daemon_stop(), *84*
lms_download, 58
lms_download(), *7, 8*, *11, 12*, *17, 18*, *28, 29*, *36, 37*, *43, 44*, *51, 52*, *60, 61*, *63–65*, *68*, *70*, *74–77*, *86*, *89, 90*, *92–94*
lms_download_status, 62
lms_download_status(), *7, 8*, *11, 12*, *17, 18*, *28, 29*, *36, 37*, *43, 44*, *51, 52*, *60, 61*, *63–65*, *68*, *70*, *74–77*, *86*, *89, 90*, *92–95*
lms_embed, 66
lms_embed(), *6–8*, *11, 12*, *17, 18*, *21, 22*, *28–30*, *32, 33*, *36, 37*, *43–45*, *47, 48*, *51–53*, *55, 56*, *60, 61*, *63, 64*, *68–71*, *74–76*, *86, 87*, *89, 90*, *92–95*, *97, 98*
lms_load, 72
lms_load(), *7, 8*, *11, 12*, *17, 18*, *22*, *28, 29*, *36, 37*, *43–45*, *51–53*, *60, 61*, *63–65*, *67, 68*, *70*, *73–77*, *86*, *89, 90*, *92–94*
lms_path, 77
lms_path(), *3*
lms_score_expected, 78
lms_server_ready, 79
lms_server_ready(), *7–9*, *11*, *13*, *17*, *19*, *28*, *30*, *36*, *43*, *45*, *51*, *53*, *60*, *63*, *69*, *74*, *76*, *82*, *86*, *89*, *91, 92*, *94*
lms_server_start, 81
lms_server_start(), *6*, *10*, *16*, *28*, *36*, *43*,

51, *59*, *63*, *68*, *74*, *80*, *86*, *89*, *92*
lms_server_status, 83
lms_server_status(), *80*
lms_server_stop, 84
lms_server_stop(), *57*
lms_unload, 85, *91*
lms_unload(), *7*, *11*, *17*, *28*, *36*, *43*, *51*, *60*, *63*, *67*, *68*, *74*, *86*, *88*, *89*, *92*
lms_unload_all, 88
lms_unload_all(), *7*, *8*, *11*, *12*, *17*, *18*, *28*, *29*, *36*, *37*, *43–45*, *51–53*, *60*, *61*, *63*, *64*, *68*, *70*, *74–76*, *86*, *89*, *90*, *92–94*

rlmstudio-conditions, 91, *99*
rlmstudio.token *(*rlmstudio_token*)*, 98
rlmstudio_api_error *(*rlmstudio-conditions*)*, 91
RLMSTUDIO_API_TOKEN *(*rlmstudio_token*)*, 98
rlmstudio_bad_response *(*rlmstudio-conditions*)*, 91
rlmstudio_context_above_max *(*lms_load*)*, 72
rlmstudio_model_mismatch *(*rlmstudio-conditions*)*, 91
rlmstudio_no_load_config *(*lms_load*)*, 72
rlmstudio_no_logprobs *(*lms_chat_openai*)*, 38
rlmstudio_no_server *(*rlmstudio-conditions*)*, 91
rlmstudio_reply_cut_off *(*rlmstudio-conditions*)*, 91
rlmstudio_token, *5*, *10*, *15*, *25*, *35*, *42*, *50*, *59*, *62*, *67*, *73*, *79*, *81*, *85*, *88*, 98

with_lms_daemon, 100