# Package 'rlmstudio'

May 5, 2026

**Title** Access and Control LM Studio

**Version** 0.2.2

**Description** A community-maintained 'R' wrapper for the 'LM Studio' command line interface and API. Provides functions to manage the local daemon and server, download and load models, and interact with Large Language Models (LLMs).

**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, knitr, mockery, rmarkdown, testthat (>= 3.0.0), withr

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Encoding** UTF-8

**Config/roxygen2/version** 8.0.0

**NeedsCompilation** no

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

**Maintainer** Jeffrey Girard <me@jmgirard.com>

**Repository** CRAN

**Date/Publication** 2026-05-05 18:30:02 UTC

## Contents

---

`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.

## Examples

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

## End(Not run)
```

## has_lms *Check if LM Studio CLI is installed*

### Description

Check if LM Studio CLI is installed

### Usage

```
has_lms()
```

### Value

A logical scalar: `TRUE` if the `lms` executable is found on the system path, and `FALSE` otherwise.

### 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). |

### 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_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 = FALSE,
  host = "http://localhost:1234"
)
```

## Arguments

| | |
|---|---|
| `loaded` | Logical. If `TRUE`, returns only currently loaded models. Defaults to `FALSE`. |
| `type` | Character vector. The types of models to include. Defaults to `c("llm", "embedding")`. |
| `detailed` | Logical. Show all information about each model. Defaults to `FALSE`. |
| `quiet` | Logical. If `TRUE`, suppresses informative console messages. Defaults to `FALSE`. |
| `host` | Character. The host address of the local server. |

## 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.

## See Also

LM Studio List Models API

## Examples

```
## Not run:
lms_server_start()
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,
  ...
)
```

## Arguments

| | |
|---|---|
| `model` | Character. The name of the loaded model. |
| `input` | Character. The user prompt to send to the model. |
| `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 | Logical. Whether to return the log probabilities of the generated tokens. Default is FALSE. |
| simplify | Logical. If TRUE, extracts the core text response. Default is TRUE. |
| ... | Additional arguments passed to the selected API body. |

## 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.
- If simplify = TRUE and logprobs = TRUE (and the chosen API type supports it), returns an object of class lms_chat_result containing both the text and a data.frame of token probabilities.

---

# 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 = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| model | Character. The loaded model name. |
| inputs | Character vector. The prompts to process. |
| system_prompt | Character. Optional system prompt. |
| format | Character. Output format: "vector", "list", or "data.frame". |
| host | Character. Server URL. |
| simplify | Logical. If TRUE, parses outputs. |
| quiet | Logical. Whether to suppress the progress bar. |
| ... | Additional arguments passed to lms_chat. |

### Value

The return type depends on the `format` argument:

- `"vector"`: A character vector of responses. This format is only supported if `simplify = TRUE` and `logprobs = FALSE`.
- `"list"`: A list where each element is the response corresponding to the provided input.
- `"data.frame"`: A data.frame containing `input` and `output` columns. If `logprobs = TRUE`, an additional list-column named `logprobs` is included.

---

## `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,
  ...
)
```

### Arguments

| | |
|---|---|
| `model` | Character. The loaded model name. |
| `input` | Character. The user prompt. |
| `system_prompt` | Character. Optional system prompt. |
| `host` | Character. Server URL. |
| `simplify` | Logical. If TRUE, parses output to text. |
| `...` | Additional API arguments. |

### Value

If `simplify = FALSE`, returns a list representing the raw JSON response. If `simplify = TRUE`, returns a character string containing the model's text output.

## 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,
  ...
)
```

### Arguments

| | |
|---|---|
| model | Character. The loaded model name. |
| messages | List. A structured list of role and content pairs. |
| host | Character. Server URL. |
| logprobs | Logical. Whether to request logprobs (currently stubbed by LM Studio). |
| simplify | Logical. If TRUE, parses output to text. |
| ... | Additional API arguments. |

### Value

If simplify = FALSE, returns a list representing the raw JSON response. Otherwise, returns a character string containing the generated text. If logprobs = TRUE, it returns an lms_chat_result object with the log probabilities populated as NULL since they are currently stubbed in the LM Studio OpenAI endpoint.

## 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,
  ...
)
```

### Arguments

| | |
|---|---|
| model | Character. The loaded model name. |
| input | Character. The user prompt. |
| instructions | Character. Optional system instructions. |
| host | Character. Server URL. |
| logprobs | Logical. Whether to return token probabilities. |
| simplify | Logical. If TRUE, parses output to text and dataframe. If FALSE, returns raw list. |
| ... | Additional API arguments (e.g., top_logprobs, temperature). |

### Value

If `simplify = FALSE`, returns a list representing the raw JSON response. Otherwise, returns a character string containing the generated text. If `logprobs = TRUE`, returns an object of class `lms_chat_result` incorporating both the text and probability data.

---

## `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()
```

### Value

Invisibly returns the process object (or 0 if already running).

### 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` | Logical. 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`. |

### Value

Invisibly returns the system exit code (0 for success).

### Desktop Users

If the daemon is currently being managed by the LM Studio desktop application, this function will fail. The CLI intentionally prevents programmatic shutdowns of the GUI to avoid disrupting visual sessions. 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", ...)
```

### Arguments

| | |
|---|---|
| model | Character. The model to download. Accepts model catalog identifiers (e.g., "openai/gpt-oss-20b") and exact Hugging Face links. |
| 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. |

### Value

A character string containing the download job_id, or "already_downloaded" if already downloaded.

### See Also

LM Studio Download Model API

### Examples

```
## Not run:
lms_server_start()

# 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")
```

### Arguments

| | |
|---|---|
| job_id | Character. The unique identifier for the download job. |
| host | Character. The host address of the local server. Defaults to "http://localhost:1234". |

## Value

An object of class `lms_download_status` containing the download status.

## See Also

LM Studio Download Status API

## Examples

```
## Not run:
lms_server_start()

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

## 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",
  ...
)
```

## Arguments

| | |
|---|---|
| `model` | Character. Unique identifier for the model to load. |
| `context_length` | Integer. Maximum number of tokens that the model will consider. |
| `eval_batch_size` | Integer. Number of input tokens to process together in a single batch during evaluation. |

`flash_attention`
: Logical. Whether to optimize attention computation.

`num_experts`
: Integer. Number of experts to use during inference for MoE models.

`offload_kv_cache_to_gpu`
: Logical. Whether KV cache is offloaded to GPU memory.

`echo_load_config`
: Logical. If `TRUE`, echoes the final load configuration in the response.

`force`
: Logical. 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`.

`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).

## Value

Invisibly returns a character string of the loaded model identifier upon success. If `echo_load_config = TRUE`, it instead invisibly returns a list containing the model's detailed load configuration.

## See Also

LM Studio Load Model API

## Examples

```
## Not run:
lms_server_start()
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.

## 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`). |

## Value

A named list containing three numeric elements (`expected_value`, `weighted_sd`, `entropy`) and a `data.frame` named `probabilities` with columns `label` and `prob`. Returns `NULL` if the input dataframe is empty or invalid.

## 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),
  stringsAsFactors = FALSE
)
```

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

---

`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)
```

## Arguments

| | |
|---|---|
| `port` | Integer. Port to run the server on. If not provided, LM Studio uses the last used port. |
| `cors` | Logical. Enable CORS support for web application development. Defaults to FALSE. |

## Value

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

## See Also

LM Studio CLI Server Start Documentation

## Examples

```
## Not run:
# Start server on the default port
lms_server_start()

# Start server on a custom port with CORS enabled
lms_server_start(port = 8080, cors = TRUE)

## 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` | Logical. Output the status in machine-readable JSON format. |
| `verbose` | Logical. Enable detailed logging output. |
| `quiet` | Logical. Suppress all logging output. |
| `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()

# 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()
```

### 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()
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", ...)
```

### Arguments

| | |
|---|---|
| `model` | Character. Unique identifier (`instance_id`) of the model instance to unload. |
| `host` | Character. The host address of the local server. Defaults to "http://localhost:1234". |
| `...` | Additional arguments passed to the API request body. |

### Value

Invisibly returns a character string representing the unloaded `instance_id` upon success.

### 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()
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", ...)
```

### 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. |

### Value

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

### See Also

`lms_unload`

## Examples

```
## Not run:
lms_server_start()
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)
```

---

`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`) will fail because the CLI prevents programmatic shutdowns of the graphical interface. 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