

Package ‘llm.api’

May 16, 2026

Title Minimal LLM Chat Interface

Version 0.1.3

Description A minimal-dependency client for Large Language Model chat APIs. Supports 'OpenAI' <<https://openai.com/>>, 'Anthropic' 'Claude' <<https://claude.com/>>, 'Moonshot' 'Kimi' <<https://www.moonshot.ai/>>, 'Ollama' <<https://ollama.com/>>, and other 'OpenAI'-compatible endpoints. Includes an agent loop with tool use and a 'Model Context Protocol' client <<https://modelcontextprotocol.io/>>. API design is derived from the 'ellmer' package, reimplemented with only base R, 'curl', and 'jsonlite'.

License MIT + file LICENSE

URL <https://github.com/cornball-ai/llm.api>

BugReports <https://github.com/cornball-ai/llm.api/issues>

Encoding UTF-8

Imports curl, jsonlite

Suggests tinytest

NeedsCompilation no

Author Troy Hernandez [aut, cre] (ORCID: <<https://orcid.org/0009-0005-4248-604X>>), cornball.ai [cph], ellmer team [cph] (API design derived from ellmer)

Maintainer Troy Hernandez <troy@cornball.ai>

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2026-05-15 22:00:09 UTC

Contents

agent	2
chat	3

chat_claude	5
chat_ollama	5
chat_openai	6
chat_session	7
chat_session_anthropic	8
chat_session_ollama	9
chat_session_openai	9
create_agent	10
history_count_tool_calls	11
history_tool_calls	11
list_ollama_models	12
llm_base	13
llm_key	13
mcp_call	14
mcp_close	15
mcp_connect	15
mcp_start	16
mcp_tools	17
mcp_tools_for_api	17
mcp_tools_for_claude	18
prices_snapshot_date	18
print.mcp_connection	19
provider_default_model	20

Index 21

agent	<i>Chat with tool use (agentic mode)</i>
-------	--

Description

Send a prompt to an LLM with tools. Automatically handles tool calls in a loop until the model responds with text only.

Usage

```
agent(prompt, tools = list(), tool_handler = NULL, system = NULL, model = NULL,
      provider = c("anthropic", "openai", "moonshot", "ollama"),
      max_turns = 20L, verbose = TRUE, history = NULL, ...)
```

Arguments

prompt	Character. The user message.
tools	List. Tool definitions (from mcp_tools_for_claude or manual).
tool_handler	Function. Called with (name, args), returns result string.
system	Character. System prompt.
model	Character. Model name.

provider	Character. Provider: "anthropic", "openai", "moonshot", or "ollama".
max_turns	Integer. Maximum tool-use turns (default: 20).
verbose	Logical. Print tool calls and results.
history	List or NULL. Previous conversation history to continue from.
...	Additional parameters passed to the API.

Value

List with final response and conversation history. The returned `$usage` carries cumulative `input_tokens`, `output_tokens`, `total_tokens`, and `cost` (USD scalar, derived from the bundled price snapshot; `0` for Ollama; `NA_real_` for models not in the snapshot).

Examples

```
## Not run:
# With MCP server
conn <- mcp_connect("r", "mcp_server.R")
tools <- mcp_tools_for_claude(conn)

result <- agent(
  "What files are in the current directory?",
  tools = tools,
  tool_handler = function(name, args) {
    mcp_call(conn, name, args)$text
  }
)

## End(Not run)
```

chat

Chat with an LLM

Description

Send a message to a Large Language Model and get a response.

Usage

```
chat(prompt, model = NULL, system = NULL, history = NULL, temperature = NULL,
     max_tokens = NULL,
     provider = c("auto", "openai", "anthropic", "moonshot", "ollama"),
     stream = FALSE, ...)
```

Arguments

prompt	Character. The user message to send.
model	Character. Model name (e.g., "gpt-4o", "claude-sonnet-4-6", "llama3.2").
system	Character or NULL. System prompt to set context.
history	List or NULL. Previous conversation turns.
temperature	Numeric or NULL. Sampling temperature (0-2).
max_tokens	Integer or NULL. Maximum tokens in response.
provider	Character. Provider: "auto", "openai", "anthropic", "moonshot", or "ollama".
stream	Logical. Stream the response (prints as it arrives).
...	Additional parameters passed to the API.

Value

A list with:

content	The assistant's response text
thinking	Chain-of-thought from reasoning models, or NULL. Populated from reasoning_content (DeepSeek, Moonshot Kimi, vLLM, SGLang), reasoning (OpenRouter), or Anthropic thinking blocks. Normalized across providers.
finish_reason	Why generation stopped. "stop" on a normal completion, "length" when truncated by max_tokens. A reasoning model that returns empty content with finish_reason == "length" ran out of budget mid-thought; raise max_tokens.
model	Model used
usage	Token usage (if available). When the model is in the bundled price snapshot, also carries cost as a USD scalar; Ollama is treated as free (cost = 0); unknown models leave cost = NA_real_. See prices_snapshot_date .
history	Updated conversation history

Examples

```
## Not run:
# Simple chat
chat("What is 2+2?")

# With system prompt
chat("Explain R", system = "You are a helpful programming tutor.")

# Continue conversation
result <- chat("Hello")
chat("Tell me more", history = result$history)

## End(Not run)
```

chat_claude	<i>Chat with Anthropic Claude</i>
-------------	-----------------------------------

Description

Convenience wrapper for 'Anthropic' 'Claude' models.

Usage

```
chat_claude(prompt, model = "claude-sonnet-4-6", ...)
```

Arguments

prompt	Character. The user message to send.
model	Character. Model name (e.g., "gpt-4o", "claude-sonnet-4-6", "llama3.2").
...	Additional parameters passed to the API.

Value

The assistant's response as a character string, or a list when history is in use. See [chat](#) for details.

Examples

```
## Not run:  
chat_claude("Explain the theory of relativity")  
chat_claude("Write a poem", model = "claude-haiku-4-5")  
  
## End(Not run)
```

chat_ollama	<i>Chat with Ollama</i>
-------------	-------------------------

Description

Convenience wrapper for local 'Ollama' models.

Usage

```
chat_ollama(prompt, model = "llama3.2", ...)
```

Arguments

prompt	Character. The user message to send.
model	Character. Model name (e.g., "gpt-4o", "claude-sonnet-4-6", "llama3.2").
...	Additional parameters passed to the API.

Value

The assistant's response as a character string, or a list when history is in use. See [chat](#) for details.

Examples

```
## Not run:
chat_ollama("What is machine learning?")
chat_ollama("Explain Docker", model = "mistral")

## End(Not run)
```

chat_openai

Chat with OpenAI

Description

Convenience wrapper for 'OpenAI' models.

Usage

```
chat_openai(prompt, model = "gpt-4o-mini", ...)
```

Arguments

prompt	Character. The user message to send.
model	Character. Model name (e.g., "gpt-4o", "claude-sonnet-4-6", "llama3.2").
...	Additional parameters passed to the API.

Value

The assistant's response as a character string, or a list when history is in use. See [chat](#) for details.

Examples

```
## Not run:
chat_openai("Explain quantum computing")
chat_openai("Write a haiku", model = "gpt-4o")

## End(Not run)
```

chat_session	<i>Create a stateful chat session</i>
--------------	---------------------------------------

Description

Creates a chat session that maintains conversation history internally. Returns a list of functions for interacting with the session.

Usage

```
chat_session(model = NULL, system_prompt = NULL,  
             provider = c("openai", "anthropic", "moonshot", "ollama"), ...)
```

Arguments

model	Character. Model name.
system_prompt	Character or NULL. System prompt.
provider	Character. Provider: "openai", "anthropic", "moonshot", or "ollama".
...	Additional parameters passed to chat().

Value

A list with functions:

chat(message)	Send a message, returns response text
stream(message)	Stream a response, returns response text
stream_async(message)	Async stream for shinychat (returns string)
last_turn()	Get the last response as list(role, text)
history()	Get full conversation history
clear()	Clear conversation history

Examples

```
## Not run:  
# Create a session  
session <- chat_session(model = "gpt-4o", system_prompt = "You are helpful.")  
  
# Chat (history maintained automatically)  
response <- session$chat("Hello")  
response2 <- session$chat("Tell me more")  
  
# Get last response  
last <- session$last_turn()  
last$text
```

```
# Clear and start over
session$clear()

## End(Not run)
```

chat_session_anthropic
Create Anthropic chat session

Description

Convenience wrapper for chat_session with Anthropic provider.

Usage

```
chat_session_anthropic(model = "claude-sonnet-4-20250514",
                       system_prompt = NULL, ...)
```

Arguments

model	Character. Model name (default: "claude-sonnet-4-20250514").
system_prompt	Character or NULL. System prompt.
...	Additional parameters passed to chat_session().

Value

A chat session list.

Examples

```
# Construct a session (no network call yet)
session <- chat_session_anthropic(system_prompt = "You are helpful.")
## Not run:
session$chat("Hello")

## End(Not run)
```

chat_session_ollama *Create Ollama chat session*

Description

Convenience wrapper for chat_session with Ollama provider.

Usage

```
chat_session_ollama(model = "llama3.2", system_prompt = NULL, ...)
```

Arguments

model	Character. Model name (default: "llama3.2").
system_prompt	Character or NULL. System prompt.
...	Additional parameters passed to chat_session().

Value

A chat session list.

Examples

```
# Construct a session (no network call yet)
session <- chat_session_ollama(system_prompt = "You are helpful.")
## Not run:
session$chat("Hello")

## End(Not run)
```

chat_session_openai *Create OpenAI chat session*

Description

Convenience wrapper for chat_session with OpenAI provider.

Usage

```
chat_session_openai(model = "gpt-4o", system_prompt = NULL, ...)
```

Arguments

model	Character. Model name (default: "gpt-4o").
system_prompt	Character or NULL. System prompt.
...	Additional parameters passed to chat_session().

Value

A chat session list.

Examples

```
# Construct a session (no network call yet)
session <- chat_session_openai(system_prompt = "You are helpful.")
## Not run:
session$chat("Hello")

## End(Not run)
```

create_agent

Create an agent with MCP servers

Description

Convenience function that sets up MCP connections and returns a function for chatting with tools.

Usage

```
create_agent(servers = list(), system = NULL, model = NULL,
             provider = c("anthropic", "openai", "moonshot", "ollama"),
             verbose = TRUE)
```

Arguments

servers	Named list of server configs. Each can be: - 'list(port = 7850)' for already-running servers - 'list(command = "r", args = "server.R", port = 7850)' to start and connect
system	Character. Default system prompt.
model	Character. Default model.
provider	Character. Provider: "anthropic", "openai", "moonshot", or "ollama".
verbose	Logical. Print tool calls.

Value

A function that takes a prompt and returns a response.

Examples

```
## Not run:
# Connect to already-running server
chat_fn <- create_agent(
  servers = list(codeR = list(port = 7850)),
  system = "You are a helpful coding assistant."
)
```

```
# Or start server automatically
chat_fn <- create_agent(
  servers = list(
    codeR = list(command = "r", args = "mcp_server.R", port = 7850)
  )
)

result <- chat_fn("List files in current directory")

## End(Not run)
```

history_count_tool_calls

Count tool calls in a history list.

Description

Thin convenience wrapper over [history_tool_calls()]. Pass 'completed_only = TRUE' to count only calls that have matching results (i.e., to skip mid-flight calls at the end of a turn that got cut off).

Usage

```
history_count_tool_calls(history, completed_only = FALSE)
```

Arguments

history List of messages.

completed_only Logical. When 'TRUE', count only calls whose result is present in the history. Default 'FALSE'.

Value

Single integer.

history_tool_calls

Walk a history list and return paired tool-call / tool-result records.

Description

Accepts either a ‘history’ list as returned by [agent()] (in which case every entry is treated as a message) or any list-of-messages that follows the same shape. Returns a list of records, one per tool call, each with:

id Canonical call id (synthesized for Ollama responses that omit one).

name Tool name.

arguments Argument list. Parsed from JSON for OpenAI-style shapes; passed through for Anthropic.

result Tool result text, or ‘NULL’ if the call has no matching result yet.

completed ‘TRUE’ when ‘result’ is non-‘NULL’.

call_message_index 1-based index of the assistant message that issued the call.

result_message_index 1-based index of the message carrying the result, or ‘NA_integer_’ for unfinished calls.

provider_shape ‘"anthropic"’ or ‘"openai"’. Useful when a consumer needs to branch on shape (rare).

Usage

```
history_tool_calls(history)
```

Arguments

history List of messages, typically the ‘history’ element from an [agent()] return value.

Value

A list of tool-call records (possibly empty). Records are returned in the order calls were issued.

list_ollama_models *List Ollama models*

Description

List all models downloaded in Ollama.

Usage

```
list_ollama_models(base_url = "http://localhost:11434")
```

Arguments

base_url Character. Ollama server URL (default: http://localhost:11434).

Value

A data frame with model information (name, size, modified).

Examples

```
## Not run:
list_ollama_models()

## End(Not run)
```

llm_base	<i>Set LLM API Base URL</i>
----------	-----------------------------

Description

Stores a base URL in the `llm.api.api_base` option, which `chat` uses as the default endpoint.

Usage

```
llm_base(url)
```

Arguments

`url` Character. Base URL for the API endpoint.

Value

The previous value of the option, invisibly.

Examples

```
old <- llm_base("http://localhost:11434") # 'ollama'
llm_base(old) # restore
```

llm_key	<i>Set LLM API Key</i>
---------	------------------------

Description

Stores an API key in the `llm.api.api_key` option, which `chat` prefers over environment variables.

Usage

```
llm_key(key)
```

Arguments

key Character. API key for authentication.

Value

The previous value of the option, invisibly.

Examples

```
old <- llm_key("sk-not-a-real-key")
llm_key(old) # restore
```

mcp_call	<i>Call a tool on an MCP server</i>
----------	-------------------------------------

Description

Call a tool on an MCP server

Usage

```
mcp_call(conn, name, arguments = list())
```

Arguments

conn An MCP connection object.
name Character. Tool name.
arguments List. Tool arguments.

Value

Tool result (list with content and text).

Examples

```
## Not run:
conn <- mcp_connect(port = 7850)
result <- mcp_call(conn, "read_file", list(path = "README.md"))
mcp_close(conn)

## End(Not run)
```

mcp_close	<i>Close an MCP connection</i>
-----------	--------------------------------

Description

Close an MCP connection

Usage

```
mcp_close(conn)
```

Arguments

conn An MCP connection object.

Value

NULL, invisibly. Called for its side effect of closing the underlying socket.

Examples

```
## Not run:  
conn <- mcp_connect(port = 7850)  
mcp_close(conn)  
  
## End(Not run)
```

mcp_connect	<i>Connect to an MCP server</i>
-------------	---------------------------------

Description

Connects to an MCP server via TCP socket.

Usage

```
mcp_connect(host = "localhost", port, name = NULL, timeout = 30)
```

Arguments

host Character. Server hostname (default: "localhost").
port Integer. Server port.
name Character. Friendly name for this server.
timeout Numeric. Connection timeout in seconds (default: 30).

Value

An MCP connection object (list with socket and tools).

Examples

```
## Not run:
# Start server first: r mcp_server.R --port 7850
conn <- mcp_connect(port = 7850, name = "codeR")
tools <- mcp_tools(conn)
result <- mcp_call(conn, "read_file", list(path = "README.md"))
mcp_close(conn)

## End(Not run)
```

mcp_start

Start and connect to an MCP server

Description

Spawns an MCP server process and connects to it. Requires the server script to support `-port` argument.

Usage

```
mcp_start(command, args = character(), port = NULL, name = NULL,
          startup_wait = 2)
```

Arguments

command	Character. Command to run the server (e.g., "r", "Rscript").
args	Character vector. Arguments (path to server script).
port	Integer. Port for the server (default: random 7850-7899).
name	Character. Friendly name.
startup_wait	Numeric. Seconds to wait for server startup.

Value

An MCP connection object.

Examples

```
## Not run:
conn <- mcp_start("Rscript", args = "mcp_server.R", port = 7850)
mcp_close(conn)

## End(Not run)
```

mcp_tools	<i>List tools from an MCP connection</i>
-----------	--

Description

List tools from an MCP connection

Usage

```
mcp_tools(conn)
```

Arguments

conn An MCP connection object.

Value

List of tool definitions.

Examples

```
## Not run:  
conn <- mcp_connect(port = 7850)  
tools <- mcp_tools(conn)  
mcp_close(conn)  
  
## End(Not run)
```

mcp_tools_for_api	<i>Format MCP tools for LLM APIs</i>
-------------------	--------------------------------------

Description

Converts MCP tool definitions to the format used by Claude/OpenAI.

Usage

```
mcp_tools_for_api(conn)
```

Arguments

conn An MCP connection, or list of connections.

Value

List of tools in API format.

Examples

```
## Not run:
conn <- mcp_connect(port = 7850)
tools <- mcp_tools_for_api(conn)
mcp_close(conn)

## End(Not run)
```

mcp_tools_for_claude *Format MCP tools for Claude API*

Description

Wrapper for `mcp_tools_for_api`, retained for backwards compatibility.

Usage

```
mcp_tools_for_claude(conn)
```

Arguments

conn An MCP connection, or list of connections.

Value

List of tools in API format.

Examples

```
## Not run:
conn <- mcp_connect(host = "localhost", port = 7850)
tools <- mcp_tools_for_claude(conn)
mcp_close(conn)

## End(Not run)
```

prices_snapshot_date *Price snapshot date*

Description

Returns the date ('YYYY-MM-DD') the bundled price table was snapshotted from BerriAI/litellm. Use this to decide when a refresh is due.

Usage

```
prices_snapshot_date()
```

Value

Character. ISO date string.

Examples

```
## Not run:  
prices_snapshot_date()  
  
## End(Not run)
```

`print.mcp_connection` *Print an MCP Connection*

Description

S3 print method for MCP connection objects.

Usage

```
## S3 method for class 'mcp_connection'  
print(x, ...)
```

Arguments

`x` An MCP connection object.
`...` Unused.

Value

`x`, invisibly. Called for the side effect of printing a summary of the connection state and available tools.

Examples

```
## Not run:  
conn <- mcp_connect(port = 7850)  
print(conn)  
mcp_close(conn)  
  
## End(Not run)
```

`provider_default_model`*Default model for a provider*

Description

Returns the model name 'chat()' falls back to when the caller doesn't specify one. Useful for client code that wants to display the resolved model upfront (e.g., in a status line) without duplicating the lookup table.

Usage

```
provider_default_model(provider)
```

Arguments

`provider` Character. One of "openai", "anthropic", "moonshot", "ollama".

Value

Character. The default model id for that provider.

Examples

```
provider_default_model("anthropic")  
provider_default_model("moonshot")
```

Index

agent, [2](#)

chat, [3](#), [5](#), [6](#), [13](#)
chat_claude, [5](#)
chat_ollama, [5](#)
chat_openai, [6](#)
chat_session, [7](#)
chat_session_anthropic, [8](#)
chat_session_ollama, [9](#)
chat_session_openai, [9](#)
create_agent, [10](#)

history_count_tool_calls, [11](#)
history_tool_calls, [11](#)

list_ollama_models, [12](#)
llm_base, [13](#)
llm_key, [13](#)

mcp_call, [14](#)
mcp_close, [15](#)
mcp_connect, [15](#)
mcp_start, [16](#)
mcp_tools, [17](#)
mcp_tools_for_api, [17](#), [18](#)
mcp_tools_for_claude, [18](#)

prices_snapshot_date, [4](#), [18](#)
print.mcp_connection, [19](#)
provider_default_model, [20](#)