Package 'llm.api'

June 26, 2026

Title Minimal LLM Chat Interface

Version 0.1.8

Date 2026-06-26

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/>, 'OpenAI' 'Codex' subscription endpoints, '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', 'jsonlite', and 'tinyoauth'.

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, tinyoauth (>= 0.1.1)

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 2026-06-26 15:00:02 UTC

Contents

`agent` *Chat with tool use (agentic mode)*

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", "anthropic_claude", "openai", "moonshot",
                   "openai_codex", "ollama"),
      max_turns = 20L, verbose = TRUE, history = NULL, history_callback = NULL,
      cache = c("none", "5m", "1h"), thinking_budget_tokens = NULL,
      web_search = FALSE, ...)
```

Arguments

prompt	Character. The user message.
tools	List. Tool definitions (from mcp_tools_for_claude or manual).
tool_handler	Function. Called with ‘(name, args)‘ and returns a result string. If it declares a formal named ‘context‘, it also receives (by name) a read-only per-call snapshot with ‘assistant_text‘ (the model’s text for this turn), ‘agent_turn‘, ‘call_index‘, ‘call_count‘, and ‘provider‘; two-argument handlers are called unchanged.
system	Character. System prompt.
model	Character. Model name.
provider	Character. Provider: "anthropic", "anthropic_claude", "openai", "moonshot", "openai_codex", 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.
history_callback	Function or NULL. Called as history_callback(history) after each assistant message is appended and after each tool result is appended. Lets callers snapshot intermediate state so an interrupt mid-turn doesn’t lose the work that was already done. Errors raised inside the callback are swallowed so telemetry/snapshotting can’t break a turn.
cache	Character. Anthropic prompt caching for the system message: "none" (default), "5m", or "1h" ephemeral TTL. Anthropic-only; warns and degrades to "none" for other providers.
thinking_budget_tokens	Integer or NULL. Anthropic extended thinking budget; must be at least 1024 and less than max_tokens. Anthropic-only; ignored with a warning for other providers.
web_search	Enable provider-native (server-side) web search: FALSE (default), TRUE, or a list of options (allowed_domains, user_location). Server-side, so it is not gated by tool_handler; the result accumulates citations and searches across turns. Wired for "openai_codex" and "openai" (OpenAI Responses web_search tool; for "openai" the run is routed through the Responses endpoint), "anthropic", "anthropic_claude", and "moonshot" (the $web_search builtin, whose calls are handled internally rather than via tool_handler); ignored with a warning otherwise.
...	Additional parameters passed to the API.

Value

List with final response and conversation history. The returned `$usage` carries cumulative `input_tokens`, `output_tokens`, `total_tokens`, and `cost` (USD scalar, derived from the bundled price snapshot; `0` for Ollama; `NA_real_` for models not in the snapshot). It also carries cumulative cache activity: `cache_read_input_tokens` (Anthropic cache reads plus OpenAI/Moonshot cached prompt tokens), `cache_creation_input_tokens` (total Anthropic cache writes), and the per-TTL split `cache_creation$ephemeral_5m_input_tokens` / `cache_creation$ephemeral_1h_input_tokens`. Passing this `$usage` back to `usage_cost` recomputes the same `cost`.

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)
```

---

anthropic_claude_credentials

Anthropic Claude subscription (OAuth) credentials

Description

Builds a zero-argument credentials function for the `anthropic_claude` provider. Tokens are obtained, cached, and refreshed by tinyoauth (see `oauth_token_anthropic`); this returns the request headers (`Authorization` and the OAuth beta header) for the current token.

Usage

```
anthropic_claude_credentials(access_token = Sys.getenv("ANTHROPIC_CLAUDE_ACCESS_TOKEN",
                                                        ""))
```

Arguments

`access_token`	Optional access token. If omitted, read from `ANTHROPIC_CLAUDE_ACCESS_TOKEN`, then from the tinyoauth cache.

Details

The `ANTHROPIC_CLAUDE_ACCESS_TOKEN` environment variable overrides the cache when set.

Value

A zero-argument credentials function returning request headers.

`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", "anthropic_claude",
                  "moonshot", "openai_codex", "ollama"),
     stream = FALSE, cache = c("none", "5m", "1h"),
     thinking_budget_tokens = NULL, web_search = FALSE, ...)
```

Arguments

`prompt`	Character. The user message to send.
`model`	Character. Model name (e.g., "gpt-5.4-mini", "claude-sonnet-4-6", "qwen3.5:9b").
`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", "anthropic_claude", "moonshot", "openai_codex", or "ollama".
`stream`	Logical. Stream the response (prints as it arrives).
`cache`	Character. Anthropic prompt caching for the system message: `"none"` (default), `"5m"`, or `"1h"` ephemeral TTL. Anthropic-only; warns and degrades to `"none"` for other providers.
`thinking_budget_tokens`	Integer or NULL. Anthropic extended thinking budget; must be at least 1024 and less than `max_tokens`. Anthropic-only; ignored with a warning for other providers.
`web_search`	Enable provider-native (server-side) web search: `FALSE` (default), `TRUE`, or a list of options (`allowed_domains`, `user_location`). The model searches on its own when useful; the result carries `citations` and `searches`. Wired for `"openai_codex"` and `"openai"` (OpenAI Responses `web_search` tool), `"anthropic"` and `"anthropic_claude"` (Messages `web_search`), and `"moonshot"` (the `$web_search` builtin); ignored with a warning for other providers. For `"openai"`, the request is routed through the Responses endpoint so search works on the default model. Moonshot doesn't expose the query or structured citations, so its `searches` carry `query = NA` and `citations` is empty (citations are inlined in the answer text).

...	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 *Chat with Anthropic Claude*

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-5.4-mini", "claude-sonnet-4-6", "qwen3.5:9b").
...	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_claude_oauth *Chat with Claude on a subscription (OAuth)*

Description

Convenience wrapper for Claude-subscription-backed models via the OAuth token from `claude_oauth_login` (no API key required).

Usage

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

Arguments

prompt	Character. The user message to send.
model	Character. Model name (e.g., "gpt-5.4-mini", "claude-sonnet-4-6", "qwen3.5:9b").
...	Additional parameters passed to the API.

Value

The assistant's response as a list. See `chat`.

Examples

```
## Not run:
claude_oauth_login()
chat_claude_oauth("Explain the theory of relativity")

## End(Not run)
```

chat_ollama *Chat with Ollama*

Description

Convenience wrapper for local 'Ollama' models.

Usage

```
chat_ollama(prompt, model = "qwen3.5:9b", ...)
```

Arguments

prompt	Character. The user message to send.
model	Character. Model name (e.g., "gpt-5.4-mini", "claude-sonnet-4-6", "qwen3.5:9b").
...	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-5.4-mini", ...)
```

Arguments

prompt	Character. The user message to send.
model	Character. Model name (e.g., "gpt-5.4-mini", "claude-sonnet-4-6", "qwen3.5:9b").
...	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-5.4-mini")

## End(Not run)
```

`chat_openai_codex` *Chat with OpenAI Codex*

Description

Convenience wrapper for ChatGPT subscription-backed Codex models.

Usage

```
chat_openai_codex(prompt, model = "gpt-5.5", ...)
```

Arguments

`prompt`	Character. The user message to send.
`model`	Character. Model name (e.g., "gpt-5.4-mini", "claude-sonnet-4-6", "qwen3.5:9b").
`...`	Additional parameters passed to the API.

Value

The assistant's response as a list. See `chat`.

Examples

```
## Not run:
creds <- openai_codex_login()
chat_openai_codex("Write a small R function", credentials = creds)

## End(Not run)
```

chat_session *Create a stateful chat session*

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", "anthropic_claude",
                          "moonshot", "openai_codex", "ollama"),
             ...)
```

Arguments

model	Character. Model name.
system_prompt	Character or NULL. System prompt.
provider	Character. Provider: "openai", "anthropic", "anthropic_claude", "moonshot", "openai_codex", 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-5.4-mini", 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-4-6", system_prompt = NULL, ...)
```

Arguments

`model`	Character. Model name (default: "claude-sonnet-4-6").
`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 = "qwen3.5:9b", system_prompt = NULL, ...)
```

Arguments

model	Character. Model name (default: "qwen3.5:9b").
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-5.4-mini", system_prompt = NULL, ...)
```

Arguments

model	Character. Model name (default: "gpt-5.4-mini").
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)
```

`chat_session_openai_codex`

Create OpenAI Codex chat session

Description

Convenience wrapper for chat_session with the OpenAI Codex provider.

Usage

```
chat_session_openai_codex(model = "gpt-5.5", system_prompt = NULL, ...)
```

Arguments

`model`	Character. Model name (default: "gpt-5.5").
`system_prompt`	Character or NULL. System prompt.
`...`	Additional parameters passed to chat_session().

Value

A chat session list.

Examples

```
## Not run:
session <- chat_session_openai_codex()
session$chat("Hello")

## End(Not run)
```

claude_oauth_login *Log in to Claude with the subscription OAuth flow*

Description

Runs tinyoauth's Claude (Claude Code) login flow, caching the token for reuse across sessions, and returns an `anthropic_claude_credentials` callback. The login is manual-paste: open the printed URL, approve, and paste the displayed code back.

Usage

```
claude_oauth_login(open_url = interactive())
```

Arguments

`open_url`	Logical. Whether to open the authorization URL in a browser.

Value

A zero-argument credentials function, invisibly. You don't normally need it: the cached token is picked up automatically by `chat(provider = "anthropic_claude")` and `chat_claude_oauth()`.

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", "anthropic_claude", "openai",
                          "moonshot", "openai_codex", "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", "anthropic_claude", "openai", "moonshot", "openai_codex", 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:

Usage

```
history_tool_calls(history)
```

Arguments

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

Details

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).

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 *Set LLM API Base URL*

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 *Set LLM API Key*

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 *Call a tool on an MCP server*

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 *Close an MCP connection*

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 *Connect to an MCP server*

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` *List tools from an MCP connection*

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` *Format MCP tools for LLM APIs*

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)
```

openai_codex_credentials *OpenAI Codex subscription credentials*

Description

Builds a zero-argument credentials function for the OpenAI Codex provider. Tokens are obtained, cached, and refreshed by tinyoauth (see `oauth_token_openai_codex`); this returns the request headers (`Authorization` and `chatgpt-account-id`) for the current token.

Usage

```
openai_codex_credentials(access_token = Sys.getenv("OPENAI_CODEX_ACCESS_TOKEN", ""),
                         account_id = Sys.getenv("OPENAI_CODEX_ACCOUNT_ID", ""))
```

Arguments

`access_token`	Optional access token. If omitted, read from `OPENAI_CODEX_ACCESS_TOKEN`, then from the tinyoauth cache.
`account_id`	Optional ChatGPT account id. If omitted, read from `OPENAI_CODEX_ACCOUNT_ID`, then from the access-token JWT.

Details

Environment variables still override the cache when set:

- `OPENAI_CODEX_ACCESS_TOKEN`
- `OPENAI_CODEX_ACCOUNT_ID`

Value

A zero-argument credentials function returning request headers.

openai_codex_login *Log in to OpenAI Codex with a device-code flow*

Description

Runs tinyoauth's ChatGPT Codex device-login flow, caching the token for reuse across sessions, and returns an `openai_codex_credentials` callback.

Usage

```
openai_codex_login(timeout = 600, open_url = interactive())
```

Arguments

`timeout`	Maximum number of seconds to wait for login.
`open_url`	Logical. Whether to open the verification URL in a browser.

Value

A zero-argument credentials function, invisibly. You don't normally need it: the cached token is picked up automatically by `chat(provider = "openai_codex")` and friends.

`prices_snapshot_date` *Bundled price-snapshot date*

Description

'llm.api' estimates 'usage$cost' from a model-price table baked into the package at release time. This returns the date that bundled table was generated.

Usage

```
prices_snapshot_date()
```

Details

It does not contact the network, check for newer prices, or update the installed package. Use the date to display staleness warnings (see [prices_snapshot_stale()]) or to decide when the package maintainer should regenerate the snapshot for a future release.

The table is generated from BerriAI/litellm's 'model_prices_and_context_window.json' (https://github.com/BerriAI/litellm/blob/main/model_prices_and_context_window.json). Cost estimates are offline and approximate, and may differ from current provider billing. Cached-input pricing follows each provider's published model: OpenAI (https://developers.openai.com/api/docs/guides/prompt-caching), Moonshot (https://www.kimi.com/help/kimi-api/api-pricing), and Anthropic (https://platform.claude.com/docs/en/build-with-claude/prompt-caching).

Value

Character scalar in 'YYYY-MM-DD' format.

See Also

[prices_snapshot_stale()], [usage_cost()]

Examples

```
prices_snapshot_date()
```

prices_snapshot_stale *Is the bundled price snapshot stale?*

Description

Convenience wrapper over [prices_snapshot_date()] for staleness alerts, so callers don't repeat the date arithmetic. Offline only; it does not check the network for newer prices.

Usage

```
prices_snapshot_stale(max_age_days = 90)
```

Arguments

max_age_days	Numeric. Age threshold in days; default 90.

Value

'TRUE' when the bundled snapshot is older than 'max_age_days', otherwise 'FALSE'.

See Also

[prices_snapshot_date()]

Examples

```
prices_snapshot_stale()
prices_snapshot_stale(max_age_days = 30)
```

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"‘, ‘"anthropic_claude"‘, ‘"moonshot"‘, ‘"openai_codex"‘, ‘"ollama"‘.

Value

Character. The default model id for that provider.

Examples

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

`usage_cost` *Estimate the USD cost of one call's token usage*

Description

Computes the offline cost estimate for a usage object, the same value ‘chat()‘ and ‘agent()‘ attach as ‘usage$cost‘. Reads whichever shape the provider returned (Anthropic's ‘input_tokens‘ / ‘output_tokens‘, or the OpenAI-compatible ‘prompt_tokens‘ / ‘completion_tokens‘) and accounts for prompt caching: Anthropic cache writes/reads via published multipliers, OpenAI / Moonshot cache hits at the bundled per-model ‘cache_read‘ rate. OpenAI cache hits are read from ‘prompt_tokens_details$cached_tokens‘ on a raw response, falling back to a flat ‘cache_read_input_tokens‘ field as exposed by ‘agent()‘ aggregates, so an ‘agent()$usage‘ object recomputes to the same cost it already carries.

Usage

```
usage_cost(model, provider, usage)
```

Arguments

`model`	Character. Model id as sent to the provider.
`provider`	Character. "anthropic", "anthropic_claude", "openai", "moonshot", "openai_codex", or "ollama".
`usage`	A usage list as found in ‘chat()$usage‘ or ‘agent()$usage‘.

Details

Costs come from the bundled price snapshot, so they are offline, approximate, and may differ from current provider billing. See [prices_snapshot_date()].

Value

Numeric scalar (USD), or ‘NA_real_‘ when ‘usage‘ is ‘NULL‘, the model isn't in the snapshot, or cache reads can't be priced.

Examples

```
## Not run:
r <- chat("hi", model = "claude-sonnet-4-6", cache = "5m")
usage_cost("claude-sonnet-4-6", "anthropic", r$usage)

## End(Not run)
```

Index