Package 'tidyllm'

October 14, 2026

Title Tidy Integration of Large Language Models

Version 0.7.1

Description A tidy interface for integrating large language model (LLM) APIs such as 'Claude', 'OpenAI', 'Gemini', 'Mistral', and local models via 'Ollama' into R workflows. The package supports text, image, audio, video, and document interactions; a unified media interface for attaching inline files or uploading to provider file stores; batch request APIs for cost-efficient large-scale processing; and a pipeline-oriented interface for seamless integration into data workflows. Web services are available at <https://www.anthropic.com>, <https://openai.com>, <https://aistudio.google.com/>, <https://mistral.ai/> and <https://ollama.com>.

License MIT + file LICENSE

URL https://edubruell.github.io/tidyllm/

BugReports https://github.com/edubruell/tidyllm/issues

Encoding UTF-8

RoxygenNote 7.3.3

VignetteBuilder knitr

Suggests later, processx, shiny, wooldridge, promises, knitr, rmarkdown, testthat (>= 3.0.0), tidyverse, httptest2, httpuv, withr, ellmer (>= 0.4.0)

Imports S7 (>= 0.2.0), base64enc, glue, jsonlite, coro, curl, httr2 (>= 1.1.1), lubridate, purrr, rlang, stringr, grDevices, pdftools, tibble, cli, png, lifecycle

Collate 'tidyllm-package.R' 'utilites.R' 'embedding_helpers.R' 'LLMMessage.R' 'APIProvider.R' 'stream_pump.R' 'chat_pipeline.R' 'async_chat.R' 'parallel_chat.R' 'example_app.R' 'llm_message.R' 'llm_verbs.R' 'provider_capabilities.R' 'media.R' 'message_retrieval.R' 'perform_api_requests.R' 'rate_limits.R' 'tidyllm_schema.R' 'tools.R' 'websearch.R' 'api_chat_completions.R' 'api_openai.R' 'api_azure_openai.R' 'api_groq.R' 'api_mistral.R' 'api_perplexity.R' 'api_deepseek.R' 'api_voyage.R' 'api_ollama.R' 'api_claude.R' 'cli_process.R'

'api_claude_cli.R' 'api_codex_cli.R' 'api_vibe_cli.R'
'api_gemini.R' 'api_ellmer.R' 'api_openrouter.R'
'api_llamacpp.R' 'pdfbatch.R' 'zzz.R'

Depends R (>= 4.2.0)

Config/testthat/edition 3

NeedsCompilation no

Author Eduard Brüll [aut, cre],
Jia Zhang [ctb]

Maintainer Eduard Brüll <eduard.bruell@zew.de>

Repository CRAN

Date/Publication 2026-10-14 08:40:02 UTC

Contents

audio_file . . . 5
azure_openai . . . 6
azure_openai_chat . . . 6
azure_openai_embedding . . . 8
cancel_job . . . 9
cancel_openai_batch . . . 10
chat . . . 10
chat_completions_chat . . . 12
chat_ellmer . . . 13
check_azure_openai_batch . . . 14
check_batch . . . 15
check_claude_batch . . . 15
check_gemini_batch . . . 16
check_groq_batch . . . 17
check_job . . . 18
check_mistral_batch . . . 18
check_openai_batch . . . 19
claude . . . 20
claude_chat . . . 20
claude_cli . . . 23
claude_cli_chat . . . 23
claude_delete_file . . . 26
claude_file_metadata . . . 26
claude_list_files . . . 27
claude_list_models . . . 28
claude_upload_file . . . 28
claude_websearch . . . 29
codex_cli . . . 30
codex_cli_chat . . . 31
deepseek . . . 33
deepseek_chat . . . 33
deep_research . . . 35

delete_file . . . 36
df_llm_message . . . 36
ellmer . . . 37
ellmer_tool . . . 37
embed . . . 39
fetch_azure_openai_batch . . . 40
fetch_batch . . . 41
fetch_claude_batch . . . 42
fetch_gemini_batch . . . 43
fetch_groq_batch . . . 43
fetch_job . . . 44
fetch_mistral_batch . . . 45
fetch_openai_batch . . . 45
field_chr . . . 46
field_object . . . 47
file_info . . . 48
gemini . . . 48
gemini_chat . . . 49
gemini_delete_file . . . 51
gemini_embedding . . . 51
gemini_file_metadata . . . 52
gemini_list_files . . . 52
gemini_list_models . . . 53
gemini_upload_file . . . 53
get_logprobs . . . 54
get_metadata . . . 55
get_partial . . . 56
get_reply . . . 56
get_reply_data . . . 57
get_stream . . . 58
get_user_message . . . 59
groq . . . 59
groq_chat . . . 60
groq_list_models . . . 62
groq_transcribe . . . 63
img . . . 64
list_azure_openai_batches . . . 64
list_batches . . . 65
list_claude_batches . . . 65
list_files . . . 66
list_gemini_batches . . . 67
list_groq_batches . . . 67
list_hf_gguf_files . . . 68
list_mistral_batches . . . 68
list_models . . . 69
list_openai_batches . . . 70
llamacpp . . . 70
llamacpp_chat . . . 71

llamacpp_delete_model . . . 73
llamacpp_download_model . . . 73
llamacpp_embedding . . . 74
llamacpp_health . . . 75
llamacpp_list_local_models . . . 75
llamacpp_list_models . . . 76
llamacpp_rerank . . . 76
LLMMessage . . . 77
llm_message . . . 78
mistral . . . 79
mistral_chat . . . 80
mistral_embedding . . . 82
mistral_list_models . . . 82
ollama . . . 83
ollama_chat . . . 84
ollama_delete_model . . . 86
ollama_download_model . . . 87
ollama_embedding . . . 87
ollama_list_models . . . 88
openai . . . 88
openai_chat . . . 89
openai_check_research . . . 90
openai_code_interpreter . . . 91
openai_deep_research . . . 91
openai_embedding . . . 92
openai_fetch_research . . . 93
openai_list_models . . . 93
openai_websearch . . . 94
openrouter . . . 94
openrouter_chat . . . 95
openrouter_credits . . . 97
openrouter_embedding . . . 97
openrouter_generation . . . 98
openrouter_list_models . . . 99
parallel_chat . . . 99
pdf_file . . . 101
pdf_page_batch . . . 102
perplexity . . . 103
perplexity_chat . . . 104
perplexity_check_research . . . 106
perplexity_deep_research . . . 107
perplexity_fetch_research . . . 109
provider_capabilities . . . 110
rate_limit_info . . . 111
send_azure_openai_batch . . . 111
send_batch . . . 113
send_chat . . . 115
send_claude_batch . . . 117

send_gemini_batch . . . 119
send_groq_batch . . . 120
send_mistral_batch . . . 122
send_ollama_batch . . . 123
send_openai_batch . . . 125
tidyllm_example_app . . . 127
tidyllm_schema . . . 128
tidyllm_tool . . . 129
upload_file . . . 130
vibe_cli . . . 130
vibe_cli_chat . . . 131
video_file . . . 133
voyage . . . 133
voyage_embedding . . . 134
voyage_rerank . . . 135
websearch . . . 136
websearch_tool . . . 138

Index **140**

`audio_file` *Create an Audio Object*

Description

Stores a reference to a local audio file for use in multimodal messages. The file is not encoded until the message is formatted for a provider.

Usage

```
audio_file(.path)
```

Arguments

`.path`	The path to the audio file on disk.

Value

A `tidyllm_audio` object.

azure_openai *Azure OpenAI Endpoint Provider Function*

Description

The azure_openai() function acts as an interface for interacting with the Azure OpenAI API through main tidyllm verbs.

Usage

```
azure_openai(..., .called_from = NULL)
```

Arguments

...	Parameters to be passed to the Azure OpenAI API specific function, such as model configuration, input text, or API-specific options.
.called_from	An internal argument that specifies which action (e.g., chat) the function is being invoked from. This argument is automatically managed and should not be modified by the user.

Details

azure_openai() currently routes messages only to azure_openai_chat() when used with chat().

send_batch(). It dynamically routes requests to OpenAI-specific functions like azure_openai_chat() and azure_openai_embedding() based on the context of the call.

Value

The result of the requested action, depending on the specific function invoked (currently, only an updated LLMMessage object for azure_openai_chat()).

azure_openai_chat *Send LLM Messages to an Azure OpenAI Chat Completions endpoint*

Description

This function sends a message history to the Azure OpenAI Chat Completions API and returns the assistant's reply.

Usage

```
azure_openai_chat(
  .llm,
  .endpoint_url = Sys.getenv("AZURE_ENDPOINT_URL"),
  .deployment = "gpt-4o-mini",
  .api_version = "2024-08-01-preview",
  .max_completion_tokens = NULL,
  .reasoning_effort = NULL,
  .frequency_penalty = NULL,
  .logit_bias = NULL,
  .presence_penalty = NULL,
  .seed = NULL,
  .stop = NULL,
  .stream = FALSE,
  .temperature = NULL,
  .top_p = NULL,
  .timeout = 60,
  .verbose = FALSE,
  .json_schema = NULL,
  .max_tries = 3,
  .dry_run = FALSE,
  .logprobs = NULL,
  .top_logprobs = NULL,
  .tools = NULL,
  .tool_choice = NULL,
  .max_tool_rounds = 10
)
```

Arguments

`.llm`	An `LLMMessage` object containing the conversation history.
`.endpoint_url`	Base URL for the API (default: Sys.getenv("AZURE_ENDPOINT_URL")).
`.deployment`	The identifier of the model that is deployed (default: "gpt-4o-mini").
`.api_version`	Which version of the API is deployed (default: "2024-08-01-preview").
`.max_completion_tokens`	An upper bound for the number of tokens that can be generated for a completion.
`.reasoning_effort`	How long should reasoning models reason (can either be "low","medium" or "high").
`.frequency_penalty`	Number between -2.0 and 2.0. Positive values penalize new tokens based on their existing frequency.
`.logit_bias`	A named list modifying the likelihood of specified tokens appearing in the completion.
`.presence_penalty`	Number between -2.0 and 2.0. Positive values penalize new tokens based on whether they appear in the text so far.

.seed	If specified, the system will make a best effort to sample deterministically.
.stop	Up to 4 sequences where the API will stop generating further tokens.
.stream	If set to TRUE, the answer will be streamed to console as it comes (default: FALSE).
.temperature	What sampling temperature to use, between 0 and 2. Higher values make the output more random.
.top_p	An alternative to sampling with temperature, called nucleus sampling.
.timeout	Request timeout in seconds (default: 60).
.verbose	Should additional information be shown after the API call (default: FALSE).
.json_schema	A JSON schema object provided by tidyllm schema or ellmer schemata.
.max_tries	Maximum retries to perform request.
.dry_run	If TRUE, perform a dry run and return the request object (default: FALSE).
.logprobs	If TRUE, get the log probabilities of each output token (default: NULL).
.top_logprobs	If specified, get the top N log probabilities of each output token (0-5, default: NULL).
.tools	Either a single TOOL object or a list of TOOL objects representing the available functions for tool calls.
.tool_choice	A character string specifying the tool-calling behavior; valid values are "none", "auto", or "required".
.max_tool_rounds	Integer specifying the maximum number of tool use iterations (default: 10).

Value

A new LLMMessage object containing the original messages plus the assistant's response.

---

azure_openai_embedding

Generate Embeddings Using OpenAI API on Azure

Description

Generate Embeddings Using OpenAI API on Azure

Usage

```
azure_openai_embedding(
  .input,
  .deployment = "text-embedding-3-small",
  .endpoint_url = Sys.getenv("AZURE_ENDPOINT_URL"),
  .api_version = "2023-05-15",
  .truncate = TRUE,
  .timeout = 120,
  .dry_run = FALSE,
  .max_tries = 3
)
```

Arguments

.input	A character vector of texts to embed or an LLMMesssageobject
.deployment	The embedding model identifier (default: "text-embedding-3-small").
.endpoint_url	Base URL for the API (default: Sys.getenv("AZURE_ENDPOINT_URL")).
.api_version	What API-Version othe Azure OpenAI API should be used (default: "2023-05-15")
.truncate	Whether to truncate inputs to fit the model's context length (default: TRUE).
.timeout	Timeout for the API request in seconds (default: 120).
.dry_run	If TRUE, perform a dry run and return the request object.
.max_tries	Maximum retry attempts for requests (default: 3).

Value

A tibble with two columns: input and embeddings. The input column contains the texts sent to embed, and the embeddings column is a list column where each row contains an embedding vector of the sent input.

---

cancel_job *Stop a running chat*

Description

Closes the connection and marks the job cancelled. A cancelled job cannot be fetched; fetch_job() on one raises rather than waiting forever.

Usage

```
cancel_job(.job)
```

Arguments

.job	A tidyllm_chat_job from send_chat().

Value

The job, invisibly.

cancel_openai_batch *Cancel an In-Progress OpenAI Batch*

Description

This function cancels an in-progress batch created through the OpenAI API. The batch will be moved to a "cancelling" state and, eventually, "cancelled."

Usage

```
cancel_openai_batch(.batch_id, .dry_run = FALSE, .max_tries = 3, .timeout = 60)
```

Arguments

.batch_id	Character; the unique identifier for the batch to cancel.
.dry_run	Logical; if TRUE, returns the constructed request without executing it (default: FALSE).
.max_tries	Integer; maximum number of retries if the request fails (default: 3).
.timeout	Integer; request timeout in seconds (default: 60).

Value

A list containing the response from the OpenAI API about the cancellation status.

chat *Chat with a Language Model*

Description

The chat() function sends a message to a language model via a specified provider and returns the response. It routes the provided LLMMessage object to the appropriate provider-specific chat function, while allowing for the specification of common arguments applicable across different providers.

Usage

```
chat(
  .llm,
  .provider = getOption("tidyllm_chat_default"),
  .dry_run = NULL,
  .stream = NULL,
  .temperature = NULL,
  .timeout = NULL,
  .top_p = NULL,
  .max_tries = NULL,
```

```
  .model = NULL,
  .verbose = NULL,
  .json_schema = NULL,
  .tools = NULL,
  .max_tool_rounds = NULL,
  .seed = NULL,
  .stop = NULL,
  .frequency_penalty = NULL,
  .presence_penalty = NULL
)
```

Arguments

`.llm`	An `LLMMessage` object containing the message or conversation history to send to the language model.
`.provider`	A function or function call specifying the language model provider and any additional parameters. This should be a call to a provider function like `openai()`, `claude()`, etc. You can also set a default provider function via the `tidyllm_chat_default` option.
`.dry_run`	Logical; if `TRUE`, simulates the request without sending it to the provider. Useful for testing.
`.stream`	Logical; if `TRUE`, streams the response from the provider in real-time.
`.temperature`	Numeric; controls the randomness of the model's output (0 = deterministic).
`.timeout`	Numeric; the maximum time (in seconds) to wait for a response.
`.top_p`	Numeric; nucleus sampling parameter, which limits the sampling to the top cumulative probability `p`.
`.max_tries`	Integer; the maximum number of retries for failed requests.
`.model`	Character; the model identifier to use (e.g., `"gpt-4"`).
`.verbose`	Logical; if `TRUE`, prints additional information about the request and response.
`.json_schema`	List; A JSON schema object as R list to enforce the output structure
`.tools`	Either a single TOOL object or a list of TOOL objects representing the available functions for tool calls.
`.max_tool_rounds`	Integer; the maximum number of tool use iterations for multi-turn tool calling (default varies by provider).
`.seed`	Integer; sets a random seed for reproducibility.
`.stop`	Character vector; specifies sequences where the model should stop generating further tokens.
`.frequency_penalty`	Numeric; adjusts the likelihood of repeating tokens (positive values decrease repetition).
`.presence_penalty`	Numeric; adjusts the likelihood of introducing new tokens (positive values encourage novelty).

Details

The `chat()` function provides a unified interface for interacting with different language model providers. Common arguments such as `.temperature`, `.model`, and `.stream` are supported by most providers and can be passed directly to `chat()`. If a provider does not support a particular argument, an error will be raised.

Advanced provider-specific configurations can be accessed via the provider functions.

Value

An updated `LLMMessage` object containing the response from the language model.

Examples

```
## Not run:
# Basic usage with OpenAI provider
llm_message("Hello World") |>
   chat(ollama(.ollama_server = "https://my-ollama-server.de"),.model="mixtral")

   chat(mistral,.model="mixtral")

# Use streaming with Claude provider
llm_message("Tell me a story") |>
   chat(claude(),.stream=TRUE)

## End(Not run)
```

`chat_completions_chat` *Chat with any OpenAI-Compatible API Endpoint*

Description

Provider function for interacting with any server that implements the OpenAI Chat Completions wire format (vLLM, LiteLLM, Together, Anyscale, etc.). Supports the `chat()` verb only.

Usage

```
chat_completions_chat(
  .llm,
  .api_url,
  .api_key_env_var = NULL,
  .model = "default",
  ...
)

chat_completions(..., .called_from = NULL)
```

Arguments

.llm	An LLMMessage object containing the conversation history.
.api_url	Base URL for the API endpoint (required).
.api_key_env_var	Name of the environment variable holding the API key.
.model	The model identifier to use (default: "default").
...	Additional parameters passed to the underlying chat function.
.called_from	Internal routing argument; do not set manually.

Value

An updated LLMMessage object with the assistant's response appended.

---

chat_ellmer *Send LLM Messages to Ellmer Chat Object*

Description

This function converts an LLMMessage to the turns ellmer chat object. The ellmer object is cloned and its turn history is cleared internatlly. This maintains tidyllms stateless style.

Usage

```
chat_ellmer(
  .llm,
  .ellmer_chat,
  .stream = FALSE,
  .timeout = 60,
  .verbose = FALSE,
  .dry_run = FALSE,
  .tools = NULL,
  .max_tries = 3
)
```

Arguments

.llm	An LLMMessage object containing the conversation history.
.ellmer_chat	An ellmer chat object (e.g., from ellmer::chat_anthropic()).
.stream	Logical; if TRUE, streams the response piece by piece (default: FALSE).
.timeout	Request timeout in seconds (default: 60).
.verbose	If TRUE, displays additional information about the chat (default: FALSE).
.dry_run	If TRUE, returns the constructed request object without executing it (default: FALSE).
.tools	Dummy for error message if called from chat with tools (default: NULL).
.max_tries	Maximum retries to perform the request (default: 3).

Value

A new LLMMessage object containing the original messages plus the assistant's response.

`check_azure_openai_batch` *Check Batch Processing Status for Azure OpenAI Batch API*

Description

This function retrieves the processing status and other details of a specified Azure OpenAI batch ID from the Azure OpenAI Batch API.

Usage

```
check_azure_openai_batch(
  .llms = NULL,
  .endpoint_url = Sys.getenv("AZURE_ENDPOINT_URL"),
  .batch_id = NULL,
  .dry_run = FALSE,
  .max_tries = 3,
  .timeout = 60
)
```

Arguments

`.llms`	A list of LLMMessage objects.
`.endpoint_url`	Base URL for the API (default: Sys.getenv("AZURE_ENDPOINT_URL")).
`.batch_id`	A manually set batch ID.
`.dry_run`	Logical; if TRUE, returns the prepared request object without executing it (default: FALSE).
`.max_tries`	Maximum retries to perform the request (default: 3).
`.timeout`	Integer specifying the request timeout in seconds (default: 60).

Value

A tibble with information about the status of batch processing.

check_batch *Check Batch Processing Status*

Description

This function retrieves the processing status and other details of a specified batchid or a list of LLMMessage objects with batch attribute. It routes the input to the appropriate provider-specific batch API function.

Usage

```
check_batch(
  .llms,
  .provider = getOption("tidyllm_cbatch_default"),
  .dry_run = NULL,
  .max_tries = NULL,
  .timeout = NULL
)
```

Arguments

.llms	A list of LLMMessage objects or a character vector with a batch ID.
.provider	A function or function call specifying the language model provider and any additional parameters. This should be a call to a provider function like openai(), claude(), etc. You can also set a default provider function via the tidyllm_cbatch_default option.
.dry_run	Logical; if TRUE, returns the prepared request object without executing it
.max_tries	Maximum retries to perform the request
.timeout	Integer specifying the request timeout in seconds

Value

A tibble with information about the status of batch processing.

check_claude_batch *Check Batch Processing Status for Claude API*

Description

This function retrieves the processing status and other details of a specified Claude batch ID from the Claude API.

Usage

```
check_claude_batch(
  .llms = NULL,
  .batch_id = NULL,
  .api_url = "https://api.anthropic.com/",
  .dry_run = FALSE,
  .max_tries = 3,
  .timeout = 60
)
```

Arguments

.llms	A list of LLMMessage objects
.batch_id	A manually set batchid
.api_url	Character; base URL of the Claude API (default: "https://api.anthropic.com/").
.dry_run	Logical; if TRUE, returns the prepared request object without executing it (default: FALSE).
.max_tries	Maximum retries to peform request
.timeout	Integer specifying the request timeout in seconds (default: 60).

Value

A tibble with information about the status of batch processing

---

check_gemini_batch *Check the Status of a Gemini Batch Operation*

Description

Retrieves processing status and metadata for a Gemini batch operation.

Usage

```
check_gemini_batch(
  .llms = NULL,
  .batch_id = NULL,
  .timeout = 60,
  .max_tries = 3,
  .dry_run = FALSE
)
```

Arguments

.llms	(Optional) List of LLMMessage objects with a "batch_id" attribute (as returned by send_gemini_batch()).
.batch_id	(Optional) Character string: full batch operation name, e.g. "batches/xyz123". If both .llms and .batch_id are provided, .batch_id is used.
.timeout	Integer. Request timeout in seconds. Default: 60.
.max_tries	Integer. Maximum retry attempts. Default: 3.
.dry_run	Logical. If TRUE, return the request object instead of making the request (for debugging). Default: FALSE.

Details

You can supply either the .batch_id string (e.g. "batches/xyz...") **or** a list of LLMMessage objects (.llms) with a "batch_id" attribute as returned by send_gemini_batch().

Value

A tibble with the operation's metadata, including name, state, creation time, completion time, and done status.

---

check_groq_batch *Check Batch Processing Status for Groq API*

Description

This function retrieves the processing status and other details of a specified Groq batch.

Usage

```
check_groq_batch(
  .llms = NULL,
  .batch_id = NULL,
  .api_url = "https://api.groq.com/",
  .dry_run = FALSE,
  .max_tries = 3,
  .timeout = 60
)
```

Arguments

.llms	A list of LLMMessage objects with a batch_id attribute.
.batch_id	A character string with the batch ID to check.
.api_url	Character; base URL of the Groq API (default: "https://api.groq.com/").
.dry_run	Logical; if TRUE, returns the prepared request object without executing (default: FALSE).
.max_tries	Maximum retries to perform request.
.timeout	Integer specifying the request timeout in seconds (default: 60).

Value

A tibble with information about the status of batch processing.

---

check_job *Check the Status of a Job*

Description

`check_job()` reports on anything tidyllm dispatched and did not wait for: a batch from `send_batch()`, a background research job from `deep_research(.background = TRUE)`, or a chat from `send_chat()`.

Usage

```
check_job(.job, ...)
```

Arguments

`.job`	A `tidyllm_chat_job`, a `tidyllm_research_job`, or a batch object from `send_batch()`.
`...`	Additional arguments passed to the underlying function.

Details

What it costs differs by job, and knowingly so: on a batch it is an HTTP call to the provider, on a chat it is a turn of this session's event loop.

Value

Status information; type depends on `.job`.

---

check_mistral_batch *Check Batch Processing Status for Mistral Batch API*

Description

This function retrieves the processing status and other details of a specified Mistral batch ID from the Mistral Batch API.

Usage

```
check_mistral_batch(
  .llms = NULL,
  .batch_id = NULL,
  .dry_run = FALSE,
  .max_tries = 3,
  .timeout = 60
)
```

Arguments

.llms	A list of LLMMessage objects.
.batch_id	A manually set batch ID.
.dry_run	Logical; if TRUE, returns the prepared request object without executing it (default: FALSE).
.max_tries	Maximum retries to perform the request (default: 3).
.timeout	Integer specifying the request timeout in seconds (default: 60).

Value

A tibble with information about the status of batch processing.

---

check_openai_batch *Check Batch Processing Status for OpenAI Batch API*

Description

This function retrieves the processing status and other details of a specified OpenAI batch ID from the OpenAI Batch API.

Usage

```
check_openai_batch(
  .llms = NULL,
  .batch_id = NULL,
  .dry_run = FALSE,
  .max_tries = 3,
  .timeout = 60
)
```

Arguments

.llms	A list of LLMMessage objects.
.batch_id	A manually set batch ID.
.dry_run	Logical; if TRUE, returns the prepared request object without executing it (default: FALSE).
.max_tries	Maximum retries to perform the request (default: 3).
.timeout	Integer specifying the request timeout in seconds (default: 60).

Value

A tibble with information about the status of batch processing.

claude *Provider Function for Claude models on the Anthropic API*

Description

The `claude()` function acts as an interface for interacting with the Anthropic API through main `tidyllm` verbs such as `chat()`, `embed()`, and `send_batch()`. It dynamically routes requests to Claude-specific functions like `claude_chat()` and `send_claude_batch()` based on the context of the call.

Usage

```
claude(..., .called_from = NULL)
```

Arguments

`...`	Parameters to be passed to the appropriate OpenAI-specific function, such as model configuration, input text, or API-specific options.
`.called_from`	An internal argument that specifies which action (e.g., `chat`, `send_batch`) the function is being invoked from. This argument is automatically managed and should not be modified by the user.

Value

The result of the requested action, depending on the specific function invoked (e.g., an updated `LLMMessage` object for `chat()`, or a matrix for `embed()`).

claude_chat *Interact with Claude AI models via the Anthropic API*

Description

Interact with Claude AI models via the Anthropic API

Usage

```
claude_chat(
  .llm,
  .model = "claude-sonnet-5-5",
  .max_tokens = 2048,
  .temperature = NULL,
  .top_k = NULL,
  .top_p = NULL,
  .metadata = NULL,
  .stop_sequences = NULL,
```

```
  .tools = NULL,
  .json_schema = NULL,
  .file_ids = NULL,
  .api_url = "https://api.anthropic.com/",
  .verbose = FALSE,
  .max_tries = 3,
  .timeout = 60,
  .stream = FALSE,
  .dry_run = FALSE,
  .thinking = FALSE,
  .thinking_budget = 1024,
  .effort = NULL,
  .cache = FALSE,
  .max_tool_rounds = 10
)
```

Arguments

.llm	An LLMMessage object containing the conversation history and system prompt.
.model	Character string specifying the Claude model version (default: "claude-sonnet-5-5").
.max_tokens	Integer specifying the maximum number of tokens in the response (default: 1024).
.temperature	Numeric between 0 and 1 controlling response randomness. Only supported on older models; Claude Sonnet 5 and Opus 4.7 or newer reject sampling parameters.
.top_k	Integer controlling diversity by limiting the top K tokens. Only supported on older models.
.top_p	Numeric between 0 and 1 for nucleus sampling. Only supported on older models.
.metadata	List of additional metadata to include with the request.
.stop_sequences	Character vector of sequences that will halt response generation.
.tools	List of additional tools or functions the model can use.
.json_schema	A schema to enforce an output structure
.file_ids	Character; A vector of file IDs for files that were uploaded to Anthropics Servers
.api_url	Base URL for the Anthropic API (default: "https://api.anthropic.com/").
.verbose	Logical; if TRUE, displays additional information about the API call (default: FALSE).
.max_tries	Maximum retries to peform request
.timeout	Integer specifying the request timeout in seconds (default: 60).
.stream	Logical; if TRUE, streams the response piece by piece (default: FALSE).
.dry_run	Logical; if TRUE, returns the prepared request object without executing it (default: FALSE).

`.thinking` Logical; if TRUE, enables Claude's thinking mode. On Claude Sonnet 4.6, Opus 4.6 or newer this maps to adaptive thinking; on older models it uses a fixed thinking budget (default: FALSE).

`.thinking_budget` Integer specifying the maximum tokens Claude can spend on thinking (default: 1024). Must be at least 1024. Only used on older models; ignored when the model supports adaptive thinking, where `.effort` controls thinking depth instead.

`.effort` Character; one of "low", "medium", "high", "xhigh", or "max". Controls thinking depth and overall token spend on models that support the effort parameter (Claude Opus 4.5 or newer, Claude Sonnet 4.6 or newer). Default NULL uses the API default ("high").

`.cache` Logical or character; enables Anthropic prompt caching for the request. TRUE caches with the default 5-minute time to live; "1h" requests a one-hour time to live. Cache reads cost roughly a tenth of the base input price. Cache token counts are reported in `get_metadata()` in the `cached_tokens` and `cache_creation_tokens` columns, and in raw form in `api_specific` under `cache_creation_input_tokens` and `cache_read_input_tokens` (default: FALSE).

Anthropic only caches prompts above a per-model minimum, and that minimum is higher on the cheaper models: 4096 tokens on Haiku 4.5, 1024 on Sonnet 5 and Sonnet 4.6, 512 on Opus 5. Shorter prompts are not an error; they simply report zero cache tokens.

`.max_tool_rounds` Integer specifying the maximum number of tool use iterations (default: 10). Set to 1 for single-round tool use, or higher for multi-turn agentic loops.

Value

A new LLMMessage object containing the original messages plus Claude's response.

Examples

```
## Not run:
# Basic usage
msg <- llm_message("What is R programming?")
result <- claude_chat(msg)

# With adaptive thinking and effort control
result2 <- claude_chat(msg,
                .thinking = TRUE,
                .effort = "low")

## End(Not run)
```

claude_cli *Chat through a locally installed Claude CLI*

Description

`claude_cli()` routes a chat to the `claude` command line tool installed on your own machine, using the login it already has. It takes no API key. See `claude_cli_chat()` for the arguments and for what happens to the CLI's own file and shell tools.

Usage

```
claude_cli(..., .called_from = NULL)
```

Arguments

`...`	Arguments passed to the CLI chat function.
`.called_from`	Internal; the verb that dispatched here.

Value

The result of the requested action; for `chat()`, an updated `LLMMessage`.

Examples

```
## Not run:
llm_message("Explain R's S7 classes in three sentences.") |>
  chat(claude_cli())

## End(Not run)
```

claude_cli_chat *Chat with Claude through your own installed Claude CLI*

Description

`claude_cli()`, like `codex_cli()` and `vibe_cli()`, sends nothing over the network itself. It runs the `claude` command line tool that is already installed and signed in on your machine, and reads its JSON output back. There is no API key to set, and usage counts against whatever plan the CLI is logged in to rather than against an Anthropic API key.

Usage

```
claude_cli_chat(
  .llm,
  .model = NULL,
  .system_prompt_mode = "append",
  .json_schema = NULL,
  .cli_tools = FALSE,
  .stateful = FALSE,
  .session_id = NULL,
  .max_budget_usd = NULL,
  .append_system_prompt = NULL,
  .binary = "claude",
  .verbose = FALSE,
  .timeout = 300,
  .stream = FALSE,
  .dry_run = FALSE
)
```

Arguments

`.llm`	An `LLMMessage` object.
`.model`	Character; the model the CLI should use, for example "claude-sonnet-5-5" or "claude-haiku-4-5-20251001". Default NULL uses whatever the CLI is configured to use.
`.system_prompt_mode`	One of "append" or "ignore". With "append" the message's system prompt is added to the CLI's own system prompt, which is the only way the CLI accepts one. With "ignore" it is dropped.
`.json_schema`	A schema to enforce an output structure; a list, or an ellmer type object. Passed to the CLI's own `--json-schema` flag.
`.cli_tools`	Controls the CLI's own tools. FALSE, the default, removes all of them and leaves out your MCP servers, so the call behaves like an ordinary completion and cannot read files. A character vector makes exactly those built-in tools available, for example `c("Read", "Glob")`; MCP tools named there still work. TRUE hands over to the CLI's own configuration, which on a default install includes Bash, Write and Edit.
`.stateful`	Logical; if TRUE the CLI keeps the conversation on its side. The first call sends the conversation and records the session id in the metadata; later calls resume that session and send only the newest user message instead of replaying the history. Default FALSE sends the whole conversation every time, the way every other tidyllm provider does.
`.session_id`	Character; resume this CLI session explicitly. Normally left NULL, because `.stateful = TRUE` picks the id up from the message history on its own.
`.max_budget_usd`	Numeric; hand the CLI a spending ceiling for this call.

`.append_system_prompt`
: Character; extra system prompt text, added after the message's own system prompt.

`.binary`
: Character; the command to run, or a full path to it. Defaults to "claude", which is looked for on the PATH and then in the places the installers use. A GUI R session often has a shorter PATH than your terminal, so if the CLI is somewhere unusual, set it once with `options(tidyllm_claude_cli_path = "/path/to/claude")` in your `.Rprofile`, or the `TIDYLLM_CLAUDE_CLI` environment variable, rather than passing it at every call.

`.verbose`
: Logical; if TRUE, prints rate limit information after the response.

`.timeout`
: Integer; seconds to wait. On a stream this is the idle deadline between events rather than a total, so a long answer is not cut off for being long.

`.stream`
: Logical; if TRUE, prints the reply as it arrives.

`.dry_run`
: Logical; if TRUE, returns the command that would be run instead of running it.

Details

`claude_cli()` finds the CLI on the PATH, and, failing that, in the directories the installers write to. RStudio and other GUI front ends do not inherit the PATH from your shell profile, so the CLI can be perfectly installed and still invisible to `Sys.which()`; the fallback is what covers that. To point at it explicitly, set `options(tidyllm_claude_cli_path = "/path/to/claude")`.

The CLI is an agent, not a plain completion endpoint. By default it can read files, edit them and run shell commands. tidyllm turns all of that off unless you ask for it, because a call to `chat()` that quietly edits files in the working directory is not what the rest of this package does. Pass `.cli_tools` to allow specific tools back.

Value

A new `LLMMessage` object containing the original messages plus the CLI's response.

Examples

```
## Not run:
llm_message("What is R's S7 class system?") |>
  chat(claude_cli())

# Let the CLI read files, but not write or run anything
llm_message("Summarise the DESCRIPTION file in this folder.") |>
  chat(claude_cli(.cli_tools = c("Read", "Glob")))

# Keep the conversation on the CLI's side
first  <- llm_message("Start a review of my package.") |>
  chat(claude_cli(.stateful = TRUE))
second <- first |>
  llm_message("Now look at the tests.") |>
  chat(claude_cli(.stateful = TRUE))

## End(Not run)
```

claude_delete_file *Delete a File from Claude API*

Description

Deletes a specific file from the Claude API using its file ID.

Usage

```
claude_delete_file(
  .file_id,
  .api_url = "https://api.anthropic.com/",
  .timeout = 60,
  .max_tries = 3,
  .dry_run = FALSE
)
```

Arguments

.file_id	The file ID to delete.
.api_url	Base URL for the Claude API (default: "https://api.anthropic.com/").
.timeout	Request timeout in seconds (default: 60).
.max_tries	Maximum retry attempts for requests (default: 3).
.dry_run	Logical; if TRUE, returns the prepared request object without executing it.

Value

Invisibly returns NULL. Prints a confirmation message upon successful deletion.

claude_file_metadata *Retrieve Metadata for a File from Claude API*

Description

Retrieves metadata for a specific file uploaded to the Claude API.

Usage

```
claude_file_metadata(
  .file_id,
  .api_url = "https://api.anthropic.com/",
  .timeout = 60,
  .max_tries = 3,
  .dry_run = FALSE
)
```

Arguments

.file_id	The file ID to retrieve metadata for.
.api_url	Base URL for the Claude API (default: "https://api.anthropic.com/").
.timeout	Request timeout in seconds (default: 60).
.max_tries	Maximum retry attempts for requests (default: 3).
.dry_run	Logical; if TRUE, returns the prepared request object without executing it.

Value

A tibble containing metadata fields such as file_id, filename, size, and MIME type.

---

claude_list_files *List Files in Claude API*

Description

Lists metadata for files uploaded to the Claude API, supporting pagination.

Usage

```
claude_list_files(
  .limit = 20,
  .order = "desc",
  .api_url = "https://api.anthropic.com/",
  .timeout = 60,
  .max_tries = 3,
  .dry_run = FALSE
)
```

Arguments

.limit	The maximum number of files to return (default: 20).
.order	Order of results, either "asc" or "desc" (default: "desc").
.api_url	Base URL for the Claude API (default: "https://api.anthropic.com/").
.timeout	Request timeout in seconds (default: 60).
.max_tries	Maximum retry attempts for requests (default: 3).
.dry_run	Logical; if TRUE, returns the prepared request object without executing it.

Value

A tibble containing metadata for each file, including file_id, filename, size, and MIME type.

claude_list_models *List Available Models from the Anthropic Claude API*

Description

List Available Models from the Anthropic Claude API

Usage

```
claude_list_models(
  .api_url = "https://api.anthropic.com",
  .timeout = 60,
  .max_tries = 3,
  .dry_run = FALSE,
  .verbose = FALSE
)
```

Arguments

.api_url	Base URL for the API (default: "https://api.anthropic.com").
.timeout	Request timeout in seconds (default: 60).
.max_tries	Maximum number of retries for the API request (default: 3).
.dry_run	Logical; if TRUE, returns the prepared request object without executing it.
.verbose	Logical; if TRUE, prints additional information about the request.

Value

A tibble containing model information (columns include type,id, display_name, and created_at), or NULL if no models are found.

claude_upload_file *Upload a File to Claude API*

Description

Uploads a file to the Claude API and returns its metadata as a tibble.

Usage

```
claude_upload_file(
  .file_path,
  .api_url = "https://api.anthropic.com/",
  .timeout = 60,
  .max_tries = 3,
  .dry_run = FALSE
)
```

Arguments

.file_path	The local file path of the file to upload.
.api_url	Base URL for the Claude API (default: "https://api.anthropic.com/").
.timeout	Request timeout in seconds (default: 60).
.max_tries	Maximum retry attempts for requests (default: 3).
.dry_run	Logical; if TRUE, returns the prepared request object without executing it.

Value

A tibble containing metadata about the uploaded file, including its file_id, name, and size.

---

claude_websearch *Builtin Claude Web Search Tool*

Description

Returns a TOOL object for Claude's builtin web_search tool. The default tool version is `web_search_20260318`, which adds the `response_inclusion` control on top of the dynamic result filtering introduced in `web_search_20260209`.

Usage

```
claude_websearch(
  .max_uses = NULL,
  .allowed_domains = NULL,
  .blocked_domains = NULL,
  .allowed_callers = "direct",
  .response_inclusion = NULL,
  .version = "web_search_20260318"
)
```

Arguments

.max_uses	Integer; maximum number of searches the model may run per request.
.allowed_domains	Character vector of domains to restrict search results to.
.blocked_domains	Character vector of domains to exclude from search results.
.allowed_callers	Character vector of callers allowed to invoke the tool; `"direct"` (default) or `"code_execution_20260120"`. Pass `NULL` to let the API apply its own default. Ignored for `"web_search_20250305"`, which has no such field.
.response_inclusion	Character; `"full"` or `"excluded"`, controlling whether search results enter the model context. Requires `web_search_20260318` or newer.
.version	Character; the Anthropic web search tool version (default: "web_search_20260318").

Details

From `web_search_20260209` onwards the API defaults `allowed_callers` to the code execution tool, which only models with programmatic tool calling support. tidyllm therefore sends `allowed_callers = "direct"` by default, so web search works on every model. Pass `.allowed_callers = "code_execution_20260120"` to opt into the dynamic filtering path on Sonnet 4.6, Opus 4.6 or newer.

`codex_cli` *Chat through a locally installed Codex CLI*

Description

`codex_cli()` routes a chat to the `codex` command line tool installed on your own machine. It uses the login Codex already has, or your OpenAI API key with `.use_api_key = TRUE`. See `codex_cli_chat()` for the arguments and for what happens to Codex's own shell and browser tools.

Usage

```
codex_cli(..., .called_from = NULL)
```

Arguments

`...`	Arguments passed to the Codex CLI chat function.
`.called_from`	Internal; the verb that dispatched here.

Value

The result of the requested action; for `chat()`, an updated `LLMMessage`.

Examples

```
## Not run:
llm_message("Explain R's S7 classes in three sentences.") |>
  chat(codex_cli())

## End(Not run)
```

codex_cli_chat *Chat with OpenAI models through your own installed Codex CLI*

Description

`codex_cli()` runs the `codex` command line tool installed on your machine and reads its JSON output back. It uses the login Codex already has, for example a ChatGPT plan, or your OpenAI API key if you set `.use_api_key = TRUE`.

Usage

```
codex_cli_chat(
  .llm,
  .model = NULL,
  .reasoning_effort = NULL,
  .system_prompt_mode = "append",
  .json_schema = NULL,
  .cli_tools = FALSE,
  .sandbox = "read-only",
  .stateful = FALSE,
  .session_id = NULL,
  .use_api_key = FALSE,
  .binary = "codex",
  .verbose = FALSE,
  .timeout = 300,
  .stream = FALSE,
  .dry_run = FALSE
)
```

Arguments

`.llm`	An `LLMMessage` object.
`.model`	Character; the model Codex should use. Default NULL uses the model Codex is configured with. The metadata reports the model only when you set one here, because Codex does not include it in its output.
`.reasoning_effort`	Character; how much the model reasons before it answers: "minimal", "low", "medium", "high" or "xhigh". Default NULL uses Codex's own setting.
`.system_prompt_mode`	One of "append" or "ignore". With "append" the message's system prompt is passed to Codex as developer instructions, next to Codex's own instructions. With "ignore" it is dropped.
`.json_schema`	A schema to enforce an output structure; a list, or an ellmer type object. Passed to Codex's `--output-schema` flag.

.cli_tools	Logical; FALSE, the default, turns off Codex's shell, browser, plugin and web search tools, so the call behaves like an ordinary completion. TRUE hands over to Codex's own configuration.
.sandbox	Character; what commands Codex runs may do when .cli_tools = TRUE. "read-only" (the default) or "workspace-write", which lets Codex change files in the working directory.
.stateful	Logical; if TRUE, Codex keeps the conversation on its side. The first call sends the conversation and records the session id in the metadata; later calls resume that session and send only the newest user message instead of replaying the history. Default FALSE sends the whole conversation every time and saves no session files.
.session_id	Character; resume this Codex session explicitly. Normally left NULL, because .stateful = TRUE picks the id up from the message history on its own.
.use_api_key	Logical; if TRUE, passes your OPENAI_API_KEY to Codex, so the call is billed to your API key instead of the login Codex has. Default FALSE.
.binary	Character; the command to run, or a full path to it. Defaults to "codex", which is looked for on the PATH and then in the places the installers use. If R does not find it, set options(tidyllm_codex_cli_path = "/path/to/codex") in your .Rprofile, or the TIDYLLM_CODEX_CLI environment variable.
.verbose	Logical; if TRUE, prints rate limit information after the response.
.timeout	Integer; seconds to wait. On a stream this is the idle deadline between events rather than a total.
.stream	Logical; if TRUE, prints each message as Codex finishes it. Codex sends whole messages, not word by word.
.dry_run	Logical; if TRUE, returns the command that would be run instead of running it.

Details

Codex is an agent, not a plain completion endpoint. By default it can run shell commands and use a browser. tidyllm turns those tools off unless you set .cli_tools = TRUE, and even then Codex runs in its read-only sandbox unless you choose another with .sandbox.

Codex adds its own instructions to every call, so even a one-line prompt sends about 12,000 input tokens. The token counts in the metadata include them.

Value

A new LLMMessage object containing the original messages plus the reply from Codex.

Examples

```
## Not run:
llm_message("What is R's S7 class system?") |>
  chat(codex_cli())

# Pay with your OpenAI API key instead of a ChatGPT login
llm_message("Explain vapply() in two sentences.") |>
  chat(codex_cli(.use_api_key = TRUE, .reasoning_effort = "low"))
```

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

---

deepseek *Deepseek Provider Function*

Description

The deepseek() function acts as a provider interface for interacting with the Deepseek API through tidyllm's chat() verb. It dynamically routes requests to deepseek-specific function. At the moment this is only deepseek_chat()

Usage

```
deepseek(..., .called_from = NULL)
```

Arguments

...	Parameters to be passed to the appropriate Deepseek-specific function, such as model configuration, input text, or API-specific options.
.called_from	An internal argument specifying which action (e.g., chat, embed) the function is invoked from. This argument is automatically managed by the tidyllm verbs and should not be modified by the user.

Value

The result of the requested action, depending on the specific function invoked (e.g., an updated LLMMessage object for chat()).

---

deepseek_chat *Send LLM Messages to the DeepSeek Chat API*

Description

This function sends a message history to the DeepSeek Chat API and returns the assistant's reply. Currently tool calls cause problems on the DeepSeek API

Usage

```
deepseek_chat(
  .llm,
  .model = "deepseek-v4-pro",
  .thinking = NULL,
  .max_tokens = 2048,
  .temperature = NULL,
  .top_p = NULL,
  .frequency_penalty = NULL,
  .presence_penalty = NULL,
  .stop = NULL,
  .stream = FALSE,
  .logprobs = NULL,
  .top_logprobs = NULL,
  .tools = NULL,
  .tool_choice = NULL,
  .api_url = "https://api.deepseek.com/",
  .timeout = 60,
  .verbose = FALSE,
  .dry_run = FALSE,
  .max_tries = 3,
  .max_tool_rounds = 10
)
```

Arguments

`.llm`	An `LLMMessage` object containing the conversation history.
`.model`	The identifier of the model to use (default: `"deepseek-v4-pro"`). Use `"deepseek-v4-flash"` for a faster, cheaper alternative.
`.thinking`	If TRUE, enables thinking mode via `thinking: {type: "enabled"}` in the request body and captures the reasoning trace in metadata (default: NULL). Supported by all V4 models; note that `temperature`, `top_p`, `presence_penalty`, and `frequency_penalty` are ignored when thinking is active.
`.max_tokens`	The maximum number of tokens that can be generated in the response (default: 2048).
`.temperature`	Controls the randomness in the model's response. Values between 0 and 2 are allowed (optional).
`.top_p`	Nucleus sampling parameter that controls the proportion of probability mass considered (optional).
`.frequency_penalty`	Number between -2.0 and 2.0. Penalizes repeated tokens to reduce repetition (optional).
`.presence_penalty`	Number between -2.0 and 2.0. Encourages new topics by penalizing tokens that have appeared so far (optional).
`.stop`	One or more sequences where the API will stop generating further tokens (optional).

.stream	Logical; if TRUE, streams the response piece by piece (default: FALSE).
.logprobs	If TRUE, returns log probabilities of each output token (default: FALSE).
.top_logprobs	Number between 0 and 5 specifying the number of top log probabilities to return (optional).
.tools	Either a single TOOL object or a list of TOOL objects representing the available functions for tool calls.
.tool_choice	A character string specifying the tool-calling behavior; valid values are "none", "auto", or "required" (optional).
.api_url	Base URL for the DeepSeek API (default: "https://api.deepseek.com/").
.timeout	Request timeout in seconds (default: 60).
.verbose	If TRUE, displays additional information after the API call (default: FALSE).
.dry_run	If TRUE, returns the constructed request object without executing it (default: FALSE).
.max_tries	Maximum retries to perform the request (default: 3).
.max_tool_rounds	Integer specifying the maximum number of tool use iterations (default: 10). Set to 1 for single-round tool use, or higher for multi-turn agentic loops.

Value

A new LLMMessage object containing the original messages plus the assistant's response.

---

deep_research *Run Deep Research via a Provider*

Description

The deep_research() function sends a message to a provider's deep research endpoint. Currently supported: Perplexity (sonar-deep-research via async API).

Usage

```
deep_research(.llm, .provider, .background = FALSE, ...)
```

Arguments

.llm	An LLMMessage object containing the research question.
.provider	A function or function call specifying the provider (e.g., perplexity()).
.background	Logical; if TRUE, returns a tidyllm_research_job immediately (default: FALSE).
...	Additional arguments passed to the provider's deep research function.

Value

If .background = FALSE, an LLMMessage with the research reply. If .background = TRUE, a tidyllm_research_job for use with check_job()/fetch_job().

delete_file *Delete a File from a Provider's File Store*

Description

Delete a File from a Provider's File Store

Usage

```
delete_file(.provider, .file_id, ...)
```

Arguments

.provider	A provider function call.
.file_id	The file ID string to delete, or a tidyllm_file object returned by upload_file().
...	Additional provider-specific arguments.

Value

Invisibly NULL; prints a confirmation message.

df_llm_message *Convert a Data Frame to an LLMMessage Object*

Description

This function converts a data frame into an LLMMessage object representing a conversation history. The data frame must have specific columns (role and content), with each row representing a message.

Usage

```
df_llm_message(.df)
```

Arguments

.df	A data frame with at least two rows and columns role and content. The role column should contain "user", "assistant", or "system". The content column should contain the corresponding message text.

Value

An LLMMessage object representing the structured conversation.

See Also

llm_message()

Other Message Creation Utilities: llm_message()

---

ellmer *Alias for the Ellmer Provider Function*

Description

The chat_ellmer() function acts as a provider interface for interacting with ellmer chat objects through tidyllm's verb interface

Usage

```
ellmer(..., .called_from = NULL)
```

Arguments

...	Additional parameters to pass through (for consistency with other providers).
.called_from	An internal argument specifying which action (e.g., chat) the function is invoked from.

Details

This function allows you to use any ellmer chat object (e.g., from ellmer::chat_anthropic(), ellmer::chat_openai(), etc.) as a stateless backend for tidyllm. The ellmer object is cloned for each interaction to maintain tidyllm's stateless approach.

Value

A provider function that can be used with tidyllm::chat().

---

ellmer_tool *Convert an ellmer Tool to a tidyllm TOOL*

Description

Converts an ellmer ToolDef or ToolBuiltIn object to a tidyllm TOOL object, allowing seamless integration of ellmer-defined tools and builtin provider tools with tidyllm workflows.

Usage

```
ellmer_tool(.ellmer_tool)
```

Arguments

.ellmer_tool	An ellmer ToolDef object created via ellmer::tool(), or a builtin tool like ellmer::claude_tool_web_search()

Details

This function supports two types of ellmer tools:

Custom ToolDef objects: Extracts the function, description, and argument schemas from an ellmer tool and converts them to tidyllm's internal representation. Ellmer type objects are automatically converted to tidyllm field descriptors.

Builtin ToolBuiltIn objects: Converts provider-native builtin tools (like Claude's web search) to tidyllm format. For builtin tools, the tool definition is passed through as-is to the provider's API, which handles the tool execution natively.

Value

A TOOL class object that can be used with tidyllm chat() functions

Examples

```
## Not run:
library(ellmer)

# Example 1: Custom tool
tool_rnorm <- ellmer::tool(
  rnorm,
  description = "Draw numbers from a random normal distribution",
  arguments = list(
    n = ellmer::type_integer("The number of observations"),
    mean = ellmer::type_number("The mean value"),
    sd = ellmer::type_number("The standard deviation")
  )
)

tidyllm_tool_rnorm <- ellmer_tool(tool_rnorm)

llm_message("Generate 100 random numbers") |>
  chat(openai(), .tools = tidyllm_tool_rnorm)

# Example 2: Builtin tool
web_search <- ellmer_tool(ellmer::claude_tool_web_search())
llm_message("What are the latest AI developments?") |>
  chat(claude(), .tools = web_search)

## End(Not run)
```

embed *Generate text embeddings*

Description

The `embed()` function allows you to embed a text via a specified provider. It routes the input to the appropriate provider-specific embedding function.

Usage

```
embed(
  .input,
  .provider = getOption("tidyllm_embed_default"),
  .model = NULL,
  .truncate = NULL,
  .timeout = NULL,
  .dry_run = NULL,
  .max_tries = NULL
)
```

Arguments

`.input`	A character vector of texts v, a list of texts and image objects, or an `LLMMessage` object
`.provider`	A function or function call specifying the language model provider and any additional parameters. This should be a call to a provider function like `openai()`, `ollama()`, etc. You can also set a default provider function via the `tidyllm_embed_default` option.
`.model`	The embedding model to use
`.truncate`	Whether to truncate inputs to fit the model's context length
`.timeout`	Timeout for the API request in seconds
`.dry_run`	If TRUE, perform a dry run and return the request object.
`.max_tries`	Maximum retry attempts for requests

Value

A tibble with two columns: `input` and `embeddings`. The `input` column contains the texts sent to embed, and the `embeddings` column is a list column where each row contains an embedding vector of the sent input.

Examples

```
## Not run:
c("What is the meaning of life, the universe and everything?",
 "How much wood would a woodchuck chuck?",
 "How does the brain work?") |>
```

```
embed(gemini)

## End(Not run)
```

`fetch_azure_openai_batch` *Fetch Results for an Azure OpenAI Batch*

Description

This function retrieves the results of a completed Azure OpenAI batch and updates the provided list of `LLMMessage` objects with the responses. It aligns each response with the original request using the `custom_ids` generated in `send_azure_openai_batch()`.

Usage

```
fetch_azure_openai_batch(
  .llms,
  .endpoint_url = Sys.getenv("AZURE_ENDPOINT_URL"),
  .batch_id = NULL,
  .dry_run = FALSE,
  .max_tries = 3,
  .timeout = 60
)
```

Arguments

`.llms`	A list of `LLMMessage` objects that were part of the batch.
`.endpoint_url`	Base URL for the API (default: Sys.getenv("AZURE_ENDPOINT_URL")).
`.batch_id`	Character; the unique identifier for the batch. By default this is NULL and the function will attempt to use the `batch_id` attribute from `.llms`.
`.dry_run`	Logical; if `TRUE`, returns the constructed request without executing it (default: `FALSE`).
`.max_tries`	Integer; maximum number of retries if the request fails (default: `3`).
`.timeout`	Integer; request timeout in seconds (default: `60`).

Value

A list of updated `LLMMessage` objects, each with the assistant's response added if successful.

`fetch_batch` *Fetch Results from a Batch API*

Description

This function retrieves the results of a completed batch and updates the provided list of `LLMMessage` objects with the responses. It aligns each response with the original request using the `custom_ids` generated in `send_batch()`.

Usage

```
fetch_batch(
  .llms,
  .provider = getOption("tidyllm_fbatch_default"),
  .dry_run = NULL,
  .max_tries = NULL,
  .timeout = NULL
)
```

Arguments

`.llms`	A list of `LLMMessage` objects containing conversation histories.
`.provider`	A function or function call specifying the language model provider and any additional parameters. This should be a call to a provider function like `openai()`, `claude()`, etc. You can also set a default provider function via the `tidyllm_fbatch_default` option.
`.dry_run`	Logical; if `TRUE`, returns the constructed request without executing it
`.max_tries`	Integer; maximum number of retries if the request fails
`.timeout`	Integer; request timeout in seconds

Details

The function routes the input to the appropriate provider-specific batch API function.

Value

A list of updated `LLMMessage` objects, each with the assistant's response added if successful.

`fetch_claude_batch` *Fetch Results for a Claude Batch*

Description

This function retrieves the results of a completed Claude batch and updates the provided list of `LLMMessage` objects with the responses. It aligns each response with the original request using the `custom_id`s generated in `send_claude_batch()`.

Usage

```
fetch_claude_batch(
  .llms,
  .batch_id = NULL,
  .api_url = "https://api.anthropic.com/",
  .dry_run = FALSE,
  .max_tries = 3,
  .timeout = 60
)
```

Arguments

`.llms`	A list of `LLMMessage` objects that were part of the batch. The list should have names (custom IDs) set by `send_claude_batch()` to ensure correct alignment.
`.batch_id`	Character; the unique identifier for the batch. By default this is NULL and the function will attempt to use the `batch_id` attribute from `.llms`.
`.api_url`	Character; the base URL for the Claude API (default: "https://api.anthropic.com/").
`.dry_run`	Logical; if `TRUE`, returns the constructed request without executing it (default: `FALSE`).
`.max_tries`	Integer; maximum number of retries if the request fails (default: `3`).
`.timeout`	Integer; request timeout in seconds (default: `60`).

Value

A list of updated `LLMMessage` objects, each with the assistant's response added if successful.

fetch_gemini_batch *Fetch Results for a Gemini Batch*

Description

Retrieves the results of a completed Gemini batch and updates the provided list of LLMMessage objects with the assistant's responses, matching by original list order.

Usage

```
fetch_gemini_batch(
  .llms,
  .batch_name = NULL,
  .timeout = 60,
  .max_tries = 3,
  .dry_run = FALSE
)
```

Arguments

.llms	List of LLMMessage objects (as from send_gemini_batch()), must have a batch_id attribute if .batch_name is not given.
.batch_name	(Optional) Character; batch operation name (e.g. "batches/xyz123"). If not provided, is taken from attr(.llms, "batch_id").
.timeout	Integer; request timeout in seconds (default: 60).
.max_tries	Integer; maximum retry attempts (default: 3).
.dry_run	Logical; if TRUE, returns the GET request object (default: FALSE).

Value

A list of updated LLMMessage objects with the assistant response appended to each, in the same order.

fetch_groq_batch *Fetch Results for a Groq Batch*

Description

This function retrieves the results of a completed Groq batch and updates the provided list of LLMMessage objects with the responses.

Usage

```
fetch_groq_batch(
  .llms,
  .batch_id = NULL,
  .api_url = "https://api.groq.com/",
  .dry_run = FALSE,
  .max_tries = 3,
  .timeout = 60
)
```

Arguments

.llms	A list of LLMMessage objects that were part of the batch.
.batch_id	Character; the unique identifier for the batch.
.api_url	Character; the base URL for the Groq API (default: "https://api.groq.com/").
.dry_run	Logical; if TRUE, returns the constructed request without executing it (default: FALSE).
.max_tries	Integer; maximum number of retries if the request fails (default: 3).
.timeout	Integer; request timeout in seconds (default: 60).

Value

A list of updated LLMMessage objects, each with the assistant's response added if successful.

---

fetch_job *Fetch the Results of a Job*

Description

fetch_job() collects what check_job() reports on: the messages of a finished batch, the report of a background research job, or the LLMMessage of a chat from send_chat(). On a chat it waits for the reply if it has not arrived yet.

Usage

```
fetch_job(.job, .provider = NULL, ...)
```

Arguments

.job	A tidyllm_chat_job, a tidyllm_research_job, or a batch object from send_batch().
.provider	A provider function; required for batch jobs and ignored by the others.
...	Additional arguments passed to the underlying function.

Value

Fetched results; type depends on .job.

fetch_mistral_batch *Fetch Results for an Mistral Batch*

Description

This function retrieves the results of a completed Mistral batch and updates the provided list of LLMMessage objects with the responses. It aligns each response with the original request using the custom_ids generated in send_mistral_batch().

Usage

```
fetch_mistral_batch(
  .llms,
  .batch_id = NULL,
  .dry_run = FALSE,
  .max_tries = 3,
  .timeout = 60
)
```

Arguments

.llms	A list of LLMMessage objects that were part of the batch.
.batch_id	Character; the unique identifier for the batch. By default this is NULL and the function will attempt to use the batch_id attribute from .llms.
.dry_run	Logical; if TRUE, returns the constructed request without executing it (default: FALSE).
.max_tries	Integer; maximum number of retries if the request fails (default: 3).
.timeout	Integer; request timeout in seconds (default: 60).

Value

A list of updated LLMMessage objects, each with the assistant's response added if successful.

fetch_openai_batch *Fetch Results for an OpenAI Batch*

Description

This function retrieves the results of a completed OpenAI batch and updates the provided list of LLMMessage objects with the responses. It aligns each response with the original request using the custom_ids generated in send_openai_batch().

Usage

```
fetch_openai_batch(
  .llms,
  .batch_id = NULL,
  .dry_run = FALSE,
  .max_tries = 3,
  .timeout = 60
)
```

Arguments

.llms	A list of LLMMessage objects that were part of the batch.
.batch_id	Character; the unique identifier for the batch. By default this is NULL and the function will attempt to use the batch_id attribute from .llms.
.dry_run	Logical; if TRUE, returns the constructed request without executing it (default: FALSE).
.max_tries	Integer; maximum number of retries if the request fails (default: 3).
.timeout	Integer; request timeout in seconds (default: 60).

Value

A list of updated LLMMessage objects, each with the assistant's response added if successful.

---

field_chr *Define Field Descriptors for JSON Schema*

Description

These functions create field descriptors used in tidyllm_schema() or field_object() to define JSON schema fields. They support character, factor, numeric, and logical types.

Usage

```
field_chr(.description = character(0), .vector = FALSE)

field_fct(.description = character(0), .levels, .vector = FALSE)

field_dbl(.description = character(0), .vector = FALSE)

field_lgl(.description = character(0), .vector = FALSE)
```

Arguments

.description	A character string describing the field (optional).
.vector	A logical value indicating if the field is a vector (default: FALSE).
.levels	A character vector specifying allowable values (for field_fct() only).

Value

An S7 tidyllm_field object representing the field descriptor.

Examples

```
field_chr("A common street name")
field_fct("State abbreviation", .levels = c("CA", "TX", "Other"))
field_dbl("House number")
field_lgl("Is residential")
field_dbl("A list of appartment numbers at the address",.vector=TRUE )
```

---

field_object *Define a nested object field*

Description

Define a nested object field

Usage

```
field_object(.description = character(0), ..., .vector = FALSE)
```

Arguments

.description	A character string describing the field (optional).
...	Named fields to include in the object definition (required).
.vector	A logical value indicating if the field is a list of objects (default: FALSE).

Value

An S7 tidyllm_field object of type "object" containing nested fields.

Examples

```
# Define an address object with nested fields
address <- field_object("A mailing address",
  street = field_chr("Street name"),
  city = field_chr("City name"),
  zipcode = field_chr("Postal code")
)

# Create a vector of objects
addresses <- field_object("List of addresses",
  street = field_chr("Street name"),
  city = field_chr("City name"),
  .vector = TRUE
)
```

file_info *Get Metadata for a File Stored on a Provider*

Description

Get Metadata for a File Stored on a Provider

Usage

```
file_info(.provider, .file_id, ...)
```

Arguments

.provider	A provider function call.
.file_id	The file ID string to look up, or a tidyllm_file object returned by upload_file().
...	Additional provider-specific arguments.

Value

A single-row tibble of file metadata.

gemini *Google Gemini Provider Function*

Description

The gemini() function acts as a provider interface for interacting with the Google Gemini API through tidyllm's main verbs such as chat() and embed(). It dynamically routes requests to Gemini-specific functions like gemini_chat() and gemini_embedding() based on the context of the call.

Usage

```
gemini(..., .called_from = NULL)
```

Arguments

...	Parameters to be passed to the appropriate Gemini-specific function, such as model configuration, input text, or API-specific options.
.called_from	An internal argument specifying which action (e.g., chat, embed) the function is invoked from. This argument is automatically managed by the tidyllm verbs and should not be modified by the user.

Details

Some functions, such as `gemini_upload_file()` and `gemini_delete_file()`, are specific to Gemini and do not have general verb counterparts.

Value

The result of the requested action, depending on the specific function invoked (e.g., an updated `LLMMessage` object for `chat()`).

`gemini_chat` *Send LLMMessage to Gemini API*

Description

Send LLMMessage to Gemini API

Usage

```
gemini_chat(
  .llm,
  .model = "gemini-3.8-flash",
  .fileid = NULL,
  .temperature = NULL,
  .max_output_tokens = NULL,
  .top_p = NULL,
  .top_k = NULL,
  .grounding_threshold = NULL,
  .presence_penalty = NULL,
  .frequency_penalty = NULL,
  .stop_sequences = NULL,
  .safety_settings = NULL,
  .json_schema = NULL,
  .tools = NULL,
  .thinking_level = NULL,
  .thinking_budget = NULL,
  .timeout = 120,
  .dry_run = FALSE,
  .max_tries = 3,
  .verbose = FALSE,
  .stream = FALSE,
  .max_tool_rounds = 10
)
```

Arguments

.llm	An existing LLMMessage object or an initial text prompt.
.model	The model identifier (default: "gemini-1.5-flash").
.fileid	Optional vector of file IDs uploaded via gemini_upload_file() (default: NULL).
.temperature	Controls randomness in generation (default: NULL, range: 0.0-2.0). Gemini 3.6 Flash and later models ignore it, and Google plans to reject it on upcoming models.
.max_output_tokens	Maximum tokens in the response (default: NULL).
.top_p	Controls nucleus sampling (default: NULL, range: 0.0-1.0). Gemini 3.6 Flash and later models ignore it, and Google plans to reject it on upcoming models.
.top_k	Controls diversity in token selection (default: NULL, range: 0 or more). Gemini 3.6 Flash and later models ignore it, and Google plans to reject it on upcoming models.
.grounding_threshold	A grounding threshold between 0 and 1. With lower grounding thresholds Gemini will use Google to search for relevant information before answering. (default: NULL).
.presence_penalty	Penalizes new tokens (default: NULL, range: -2.0 to 2.0).
.frequency_penalty	Penalizes frequent tokens (default: NULL, range: -2.0 to 2.0).
.stop_sequences	Optional character sequences to stop generation (default: NULL, up to 5).
.safety_settings	A list of safety settings (default: NULL).
.json_schema	A schema to enforce an output structure
.tools	Either a single TOOL object or a list of TOOL objects representing the available functions for tool calls.
.thinking_level	How much the model reasons before answering: "minimal", "low", "medium" or "high" (default: NULL, the model's own default). Supported levels vary by model; gemini-3.8-flash accepts "low", "medium" and "high".
.thinking_budget	Deprecated. Token budget for internal reasoning (default: NULL). Use .thinking_level instead; Google plans to reject token budgets on upcoming models. Cannot be combined with .thinking_level.
.timeout	When should our connection time out (default: 120 seconds).
.dry_run	If TRUE, perform a dry run and return the request object.
.max_tries	Maximum retries to perform request (default: 3).
.verbose	Should additional information be shown after the API call.
.stream	Should the response be streamed (default: FALSE).
.max_tool_rounds	Integer specifying the maximum number of tool use iterations (default: 10). Set to 1 for single-round tool use, or higher for multi-turn agentic loops.

Value

A new LLMMessage object containing the original messages plus the assistant's response.

---

gemini_delete_file *Delete a File from Gemini API*

Description

Deletes a specific file from the Gemini API using its file ID.

Usage

```
gemini_delete_file(.file_name)
```

Arguments

.file_name	The file ID (e.g., "files/abc-123") to delete.

Value

Invisibly returns NULL. Prints a confirmation message upon successful deletion.

---

gemini_embedding *Generate Embeddings Using the Google Gemini API*

Description

Generate Embeddings Using the Google Gemini API

Usage

```
gemini_embedding(
  .input,
  .model = "gemini-embedding-2",
  .truncate = TRUE,
  .timeout = 120,
  .dry_run = FALSE,
  .max_tries = 3
)
```

Arguments

.input	A character vector of texts to embed or an LLMMessage object
.model	The embedding model identifier (default: "gemini-embedding-2").
.truncate	Whether to truncate inputs to fit the model's context length (default: TRUE).
.timeout	Timeout for the API request in seconds (default: 120).
.dry_run	If TRUE, perform a dry run and return the request object.
.max_tries	Maximum retry attempts for requests (default: 3).

Value

A matrix where each column corresponds to the embedding of a message in the message history.

---

gemini_file_metadata *Retrieve Metadata for a File from Gemini API*

Description

Retrieves metadata for a specific file uploaded to the Gemini API.

Usage

```
gemini_file_metadata(.file_name)
```

Arguments

.file_name	The file ID (e.g., "files/abc-123") to retrieve metadata for.

Value

A tibble containing metadata fields such as name, display name, MIME type, size, and URI.

---

gemini_list_files *List Files in Gemini API*

Description

Lists metadata for files uploaded to the Gemini API, supporting pagination.

Usage

```
gemini_list_files(.page_size = 10, .page_token = NULL)
```

Arguments

.page_size	The maximum number of files to return per page (default: 10, maximum: 100).
.page_token	A token for fetching the next page of results (default: NULL).

Value

A tibble containing metadata for each file, including fields such as name, display name, MIME type, and URI.

---

gemini_list_models *List Available Models from the Google Gemini API*

Description

List Available Models from the Google Gemini API

Usage

```
gemini_list_models(.timeout = 60, .max_tries = 3, .dry_run = FALSE)
```

Arguments

.timeout	Request timeout in seconds (default: 60).
.max_tries	Maximum number of retries for the API request (default: 3).
.dry_run	Logical; if TRUE, returns the prepared request object without executing it.

Value

A tibble containing model information with columns including `name`, `base_model_id`, `version`, `display_name`, `description`, `input_token_limit`, `output_token_limit`, `supported_generation_methods`, `thinking`, `temperature`, `max_temperature`, `top_p`, and `top_k`, or NULL if no models are found.

---

gemini_upload_file *Upload a File to Gemini API*

Description

Uploads a file to the Gemini API and returns its metadata as a tibble.

Usage

```
gemini_upload_file(.file_path)
```

Arguments

`.file_path`	The local file path of the file to upload.

Value

A tibble containing metadata about the uploaded file, including its name, URI, and MIME type.

`get_logprobs` *Retrieve Log Probabilities from Assistant Replies*

Description

Extracts token log probabilities from assistant replies within an `LLMMessage` object. Each row represents a token with its log probability and top alternative tokens.

Usage

```
get_logprobs(.llm, .index = NULL)
```

Arguments

`.llm`	An `LLMMessage` object containing the message history.
`.index`	A positive integer specifying which assistant reply's log probabilities to extract. If `NULL` (default), log probabilities for all replies are returned.

Details

An empty tibble is output if no logprobs were requested. Works with `openai_chat()`, `llamacpp_chat()`, and other providers that support logprobs.

Columns include:

- `reply_index`: The index of the assistant reply in the message history.
- `token`: The generated token.
- `logprob`: The log probability of the generated token.
- `bytes`: The byte-level encoding of the token.
- `top_logprobs`: A list column containing the top alternative tokens with their log probabilities.

Value

A tibble containing log probabilities for the specified assistant reply or all replies.

See Also

`get_metadata()`

get_metadata *Retrieve Metadata from Assistant Replies*

Description

Retrieves metadata from assistant replies within an LLMMessage object. It returns the metadata as a tibble.

Usage

```
get_metadata(.llm, .index = NULL)

last_metadata(.llm)
```

Arguments

.llm	An LLMMessage object containing the message history.
.index	A positive integer specifying which assistant reply's metadata to extract. If NULL (default), metadata for all replies is returned.

Details

Metadata columns may include:

- model: The model used for generating the reply.
- timestamp: The time when the reply was generated.
- prompt_tokens: The number of tokens in the input prompt.
- completion_tokens: The number of tokens in the assistant's reply.
- total_tokens: The total number of tokens (prompt + completion).
- cached_tokens: Prompt tokens served from the provider's prompt cache. NA_integer_ for providers that do not report cache usage; 0 means a reported cache miss.
- cache_creation_tokens: Prompt tokens written to the cache. Currently reported by claude() only; NA_integer_ elsewhere.
- api_specific: A list column with API-specific metadata, including the raw provider-specific cache fields.

For convenience, last_metadata() is provided to retrieve the metadata for the last message.

Value

A tibble containing metadata for the specified assistant reply or all replies.

See Also

last_metadata()

get_partial *The text a running chat has produced so far*

Description

Non-blocking: it advances the job as far as it can right now and returns what has arrived. On a finished job it returns the complete text, so a polling loop needs no special case at the end. On a non-streaming job it returns "" until the job is done, because there is nothing to report before then.

Usage

```
get_partial(.job)
```

Arguments

.job	A tidyllm_chat_job from send_chat().

Value

A character scalar.

get_reply *Retrieve Assistant Reply as Text*

Description

Extracts the plain text content of the assistant's reply from an LLMMessage object. Use get_reply_data() for structured replies in JSON format.

Usage

```
get_reply(.llm, .index = NULL)

last_reply(.llm)
```

Arguments

.llm	An LLMMessage object containing the message history.
.index	A positive integer indicating the index of the assistant reply to retrieve. Defaults to NULL, which retrieves the last reply.

Details

This function is the core utility for retrieving assistant replies by index. For convenience, last_reply() is provided as a wrapper to retrieve the latest assistant reply.

Value

Returns a character string containing the assistant's reply, or `NA_character_` if no reply exists.

See Also

`get_reply_data()`, `last_reply()`

`get_reply_data` *Retrieve Assistant Reply as Structured Data*

Description

Parses the assistant's reply as JSON and returns the corresponding structured data. If the reply is not marked as JSON, attempts to extract and parse JSON content from the text.

Usage

```
get_reply_data(.llm, .index = NULL)

last_reply_data(.llm)
```

Arguments

`.llm`	An `LLMMessage` object containing the message history.
`.index`	A positive integer indicating the index of the assistant reply to retrieve. Defaults to `NULL`, which retrieves the last reply.

Details

For convenience, `last_reply_data()` is provided as a wrapper to retrieve the latest assistant reply's data.

Value

Returns the parsed data from the assistant's reply, or `NULL` if parsing fails.

See Also

`get_reply()`, `last_reply_data()`

get_stream *A running chat's deltas, as a stream*

Description

Returns a coro async generator over the text deltas: the pull form of .on_chunk. It replays whatever has already arrived and then continues live, so it is safe to ask for at any point in a job's life and always tells the same story as get_partial().

Usage

```
get_stream(.job)
```

Arguments

.job	A streaming tidyllm_chat_job from send_chat().

Details

This is what shinychat::chat_append() consumes directly, because shinychat's whole extensibility contract is inherits(x, "coro_generator_instance"). The generator is asynchronous rather than synchronous on purpose: shinychat's consumer only yields the event loop at an await(), so a synchronous generator would drain in one tick and the reply would appear all at once instead of token by token.

In a plain script this is rarely what you want. An async generator hands a coro::loop() promises rather than text, so consuming it takes coro::async() and coro::await_each(); .on_chunk or a get_partial() loop says the same thing with less ceremony.

Value

A coro async generator instance yielding character deltas.

Examples

```
## Not run:
job <- llm_message("Tell me a story") |> send_chat(claude(), .stream = TRUE)
shinychat::chat_append("chat", get_stream(job))

## End(Not run)
```

get_user_message *Retrieve a User Message by Index*

Description

Extracts the content of a user's message from an LLMMessage object at a specific index.

Usage

```
get_user_message(.llm, .index = NULL)

last_user_message(.llm)
```

Arguments

.llm	An LLMMessage object.
.index	A positive integer indicating which user message to retrieve. Defaults to NULL, which retrieves the last message.

Details

For convenience, last_user_message() is provided as a wrapper to retrieve the latest user message without specifying an index.

Value

Returns the content of the user's message at the specified index. If no messages are found, returns NA_character_.

See Also

last_user_message()

groq *Groq API Provider Function*

Description

The groq() function acts as an interface for interacting with the Groq API through tidyllm's main verbs. Currently, Groq only supports groq_chat() for chat-based interactions and groq_transcribe() for transcription tasks.

Usage

```
groq(..., .called_from = NULL)
```

Arguments

...	Parameters to be passed to the Groq-specific function, such as model configuration, input text, or API-specific options.
.called_from	An internal argument that specifies which action (e.g., chat) the function is being invoked from. This argument is automatically managed and should not be modified by the user.

Details

Since groq_transcribe() is unique to Groq and does not have a general verb counterpart, groq() currently routes messages only to groq_chat() when used with verbs like chat().

Value

The result of the requested action, depending on the specific function invoked (currently, only an updated LLMMessage object for groq_chat()).

---

groq_chat *Send LLM Messages to the Groq Chat API*

Description

This function sends a message history to the Groq Chat API and returns the assistant's reply.

Usage

```
groq_chat(
  .llm,
  .model = "openai/gpt-oss-120b",
  .max_tokens = 1024,
  .temperature = NULL,
  .top_p = NULL,
  .frequency_penalty = NULL,
  .presence_penalty = NULL,
  .stop = NULL,
  .seed = NULL,
  .tools = NULL,
  .tool_choice = NULL,
  .api_url = "https://api.groq.com/",
  .json_schema = NULL,
  .timeout = 60,
  .verbose = FALSE,
  .stream = FALSE,
  .dry_run = FALSE,
  .max_tries = 3,
  .max_tool_rounds = 10
)
```

Arguments

.llm	An LLMMessage object containing the conversation history.
.model	The identifier of the model to use (default: "llama-3.2-11b-vision-preview").
.max_tokens	The maximum number of tokens that can be generated in the response (default: 1024).
.temperature	Controls the randomness in the model's response. Values between 0 and 2 are allowed, where higher values increase randomness (optional).
.top_p	Nucleus sampling parameter that controls the proportion of probability mass considered. Values between 0 and 1 are allowed (optional).
.frequency_penalty	Number between -2.0 and 2.0. Positive values penalize repeated tokens, reducing likelihood of repetition (optional).
.presence_penalty	Number between -2.0 and 2.0. Positive values encourage new topics by penalizing tokens that have appeared so far (optional).
.stop	One or more sequences where the API will stop generating further tokens. Can be a string or a list of strings (optional).
.seed	An integer for deterministic sampling. If specified, attempts to return the same result for repeated requests with identical parameters (optional).
.tools	Either a single TOOL object or a list of TOOL objects representing the available functions for tool calls (optional).
.tool_choice	A character string specifying the tool-calling behavior; valid values are "none", "auto", or "required" (optional).
.api_url	Base URL for the Groq API (default: "https://api.groq.com/").
.json_schema	A list or tidyllm schema created with tidyllm_schema() for structured JSON output (optional).
.timeout	Request timeout in seconds (default: 60).
.verbose	If TRUE, displays additional information after the API call, including rate limit details (default: FALSE).
.stream	Logical; if TRUE, streams the response piece by piece (default: FALSE).
.dry_run	If TRUE, performs a dry run and returns the constructed request object without executing it (default: FALSE).
.max_tries	Maximum retries to peform request
.max_tool_rounds	Integer specifying the maximum number of tool use iterations (default: 10). Set to 1 for single-round tool use, or higher for multi-turn agentic loops.

Value

A new LLMMessage object containing the original messages plus the assistant's response.

Examples

```
## Not run:
# Basic usage
msg <- llm_message("What is Groq?")
result <- groq_chat(msg)

# With custom parameters
result2 <- groq_chat(msg,
               .model = "llama-3.2-vision",
               .temperature = 0.5,
               .max_tokens = 512)

## End(Not run)
```

`groq_list_models` *List Available Models from the Groq API*

Description

List Available Models from the Groq API

Usage

```
groq_list_models(
  .api_url = "https://api.groq.com",
  .timeout = 60,
  .max_tries = 3,
  .dry_run = FALSE,
  .verbose = FALSE
)
```

Arguments

`.api_url`	Base URL for the API (default: "https://api.groq.com").
`.timeout`	Request timeout in seconds (default: 60).
`.max_tries`	Maximum number of retries for the API request (default: 3).
`.dry_run`	Logical; if TRUE, returns the prepared request object without executing it.
`.verbose`	Logical; if TRUE, prints additional information about the request.

Value

A tibble containing model information (columns include `id`, `created`, `owned_by`, and `context_window`), or NULL if no models are found.

groq_transcribe *Transcribe an Audio File Using Groq transcription API*

Description

This function reads an audio file and sends it to the Groq transcription API for transcription.

Usage

```
groq_transcribe(
  .audio_file,
  .model = "whisper-large-v3",
  .language = NULL,
  .prompt = NULL,
  .temperature = 0,
  .api_url = "https://api.groq.com/openai/v1/audio/transcriptions",
  .dry_run = FALSE,
  .verbose = FALSE,
  .max_tries = 3
)
```

Arguments

.audio_file	The path to the audio file (required). Supported formats include flac, mp3, mp4, mpeg, mpga, m4a, ogg, wav, or webm.
.model	The model to use for transcription (default: "whisper-large-v3").
.language	The language of the input audio, in ISO-639-1 format (optional).
.prompt	A prompt to guide the transcription style. It should match the audio language (optional).
.temperature	Sampling temperature, between 0 and 1, with higher values producing more randomness (default: 0).
.api_url	Base URL for the API (default: "https://api.groq.com/openai/v1/audio/transcriptions").
.dry_run	Logical; if TRUE, performs a dry run and returns the request object without making the API call (default: FALSE).
.verbose	Logical; if TRUE, rate limiting info is displayed after the API request (default: FALSE).
.max_tries	Maximum retries to peform request

Value

A character vector containing the transcription.

Examples

```
## Not run:
# Basic usage
groq_transcribe(.audio_file = "example.mp3")

## End(Not run)
```

`img` *Create an Image Object*

Description

This function reads an image file from disk, encodes it in base64, and returns a `tidyllm_image` object that can be used in multimodal embedding requests or attached to messages via `.media`.

Usage

```
img(.path)
```

Arguments

`.path`	The path to the image file on disk.

Value

A `tidyllm_image` object.

`list_azure_openai_batches`
List Azure OpenAI Batch Requests

Description

Retrieves batch request details from the Azure OpenAI Batch API.

Usage

```
list_azure_openai_batches(
  .endpoint_url = Sys.getenv("AZURE_ENDPOINT_URL"),
  .limit = 20,
  .max_tries = 3,
  .timeout = 60
)
```

Arguments

.endpoint_url	Base URL for the API (default: Sys.getenv("AZURE_ENDPOINT_URL")).
.limit	Maximum number of batches to retrieve (default: 20).
.max_tries	Maximum retry attempts for requests (default: 3).
.timeout	Request timeout in seconds (default: 60).

Value

A tibble with batch details: batch ID, status, creation time, expiration time, and request counts (total, completed, failed).

---

list_batches *List all Batch Requests on a Batch API*

Description

List all Batch Requests on a Batch API

Usage

```
list_batches(.provider = getOption("tidyllm_lbatch_default"))
```

Arguments

.provider	A function or function call specifying the language model provider and any additional parameters. This should be a call to a provider function like `openai()`, `claude()`, etc. You can also set a default provider function via the `tidyllm_lbatch_default` option.

Value

A tibble with information about the status of batch processing.

---

list_claude_batches *List Claude Batch Requests*

Description

Retrieves batch request details from the Claude API.

Usage

```
list_claude_batches(
  .api_url = "https://api.anthropic.com/",
  .limit = 20,
  .max_tries = 3,
  .timeout = 60
)
```

Arguments

.api_url	Base URL for the Claude API (default: "https://api.anthropic.com/").
.limit	Maximum number of batches to retrieve (default: 20).
.max_tries	Maximum retry attempts for requests (default: 3).
.timeout	Request timeout in seconds (default: 60).

Value

A tibble with batch details: batch ID, status, creation time, expiration time, and request counts (succeeded, errored, expired, canceled).

---

list_files *List Files Stored on a Provider*

Description

List Files Stored on a Provider

Usage

```
list_files(.provider, ...)
```

Arguments

.provider	A provider function call.
...	Additional provider-specific arguments.

Value

A tibble of file metadata.

list_gemini_batches *List Recent Gemini Batch Operations*

Description

Returns a tibble with recent Gemini batch operations and their metadata.

Usage

```
list_gemini_batches(
  .filter = NULL,
  .page_size = 20,
  .timeout = 60,
  .max_tries = 3,
  .dry_run = FALSE
)
```

Arguments

.filter	Optional filter expression for batch listing (see Gemini API docs).
.page_size	Integer. Maximum number of results to return. Default: 20.
.timeout	Integer. Request timeout in seconds. Default: 60.
.max_tries	Integer. Maximum retry attempts. Default: 3.
.dry_run	Logical. If TRUE, returns the request object (for debugging). Default: FALSE.

Value

A tibble with columns: name, state, done, create_time, complete_time.

list_groq_batches *List Groq Batch Requests*

Description

Retrieves batch request details from the Groq API.

Usage

```
list_groq_batches(
  .api_url = "https://api.groq.com/",
  .limit = 20,
  .max_tries = 3,
  .timeout = 60
)
```

Arguments

.api_url	Base URL for the Groq API (default: "https://api.groq.com/").
.limit	Maximum number of batches to retrieve (default: 20).
.max_tries	Maximum retry attempts for requests (default: 3).
.timeout	Request timeout in seconds (default: 60).

Value

A tibble with batch details including batch ID, status, creation time, and request counts.

---

list_hf_gguf_files *List GGUF Files Available in a Hugging Face Repository*

Description

Calls the Hugging Face Hub API to list all .gguf files in a repository, along with their sizes. Useful for choosing a quantization level before calling llamacpp_download_model().

Usage

```
list_hf_gguf_files(.repo, .timeout = 30)
```

Arguments

.repo	Hugging Face repository ID in "owner/model" format (e.g. "Qwen/Qwen3-8B-GGUF").
.timeout	Request timeout in seconds (default: 30).

Value

A tibble with columns filename, size_gb, and url.

---

list_mistral_batches *List Mistral Batch Requests*

Description

Retrieves batch request details from the OpenAI Batch API.

Usage

```
list_mistral_batches(
  .limit = 100,
  .max_tries = 3,
  .timeout = 60,
  .status = NULL,
  .created_after = NULL
)
```

Arguments

.limit	Maximum number of batches to retrieve (default: 20).
.max_tries	Maximum retry attempts for requests (default: 3).
.timeout	Request timeout in seconds (default: 60).
.status	Filter by status. (default: NULL)
.created_after	created after a string specifiying a date-time (default: NULL)

Value

A tibble with batch details for all batches fitting the request

---

list_models *List Available Models for a Provider*

Description

The list_models() function retrieves available models from the specified provider.

Usage

```
list_models(.provider = getOption("tidyllm_lmodels_default"), ...)
```

Arguments

.provider	A function or function call specifying the provider and any additional parameters. You can also set a default provider via the tidyllm_lmodels_default option.
...	Additional arguments to be passed to the provider-specific list_models function.

Value

A tibble containing model information.

list_openai_batches *List OpenAI Batch Requests*

Description

Retrieves batch request details from the OpenAI Batch API.

Usage

```
list_openai_batches(.limit = 20, .max_tries = 3, .timeout = 60)
```

Arguments

.limit	Maximum number of batches to retrieve (default: 20).
.max_tries	Maximum retry attempts for requests (default: 3).
.timeout	Request timeout in seconds (default: 60).

Value

A tibble with batch details: batch ID, status, creation time, expiration time, and request counts (total, completed, failed).

llamacpp *llama.cpp Provider Function*

Description

The `llamacpp()` provider connects tidyllm to a locally running llama.cpp server. It exposes the same verb/provider pattern as every other tidyllm provider while also offering llama.cpp-specific features: BNF grammar constraints (`.grammar`), token logprobs (`.logprobs`), and model management helpers.

The server must be started separately before calling `llamacpp()`. See `llamacpp_health()` to verify the server is running, and `llamacpp_download_model()` / `list_hf_gguf_files()` to obtain models.

Usage

```
llamacpp(..., .called_from = NULL)
```

Arguments

...	Parameters passed to the appropriate llama.cpp-specific function.
.called_from	An internal argument specifying which verb invoked this function. Managed automatically by tidyllm verbs; do not set manually.

Value

The result of the requested action (e.g., an updated `LLMMessage` for `chat()`, a tibble for `embed()` or `list_models()`).

`llamacpp_chat` *Send LLM Messages to a llama.cpp Server*

Description

Sends a message history to a local llama.cpp server using the OpenAI-compatible Chat Completions API. Supports BNF grammar constraints (a llama.cpp-specific feature that enforces output format at the token sampling level) and token logprobs for uncertainty quantification.

Usage

```
llamacpp_chat(
  .llm,
  .model = "local-model",
  .max_tokens = 1024,
  .temperature = NULL,
  .top_p = NULL,
  .stop = NULL,
  .stream = FALSE,
  .tools = NULL,
  .tool_choice = NULL,
  .json_schema = NULL,
  .grammar = NULL,
  .logprobs = FALSE,
  .top_logprobs = NULL,
  .seed = NULL,
  .thinking = NULL,
  .thinking_budget = NULL,
  .server = Sys.getenv("LLAMACPP_SERVER", "http://localhost:8080"),
  .api_key = Sys.getenv("LLAMACPP_API_KEY", ""),
  .timeout = 120,
  .verbose = FALSE,
  .dry_run = FALSE,
  .max_tries = 3,
  .max_tool_rounds = 10
)
```

Arguments

`.llm`	An `LLMMessage` object containing the conversation history.
`.model`	The model name (default: `"local-model"`). llama.cpp ignores this value and serves whatever GGUF is loaded; the default is a clear placeholder.

.max_tokens	Maximum tokens in the response (default: 1024).
.temperature	Controls randomness (0–2, optional).
.top_p	Nucleus sampling parameter (0–1, optional).
.stop	One or more stop sequences (optional).
.stream	Logical; if TRUE, streams the response (default: FALSE).
.tools	Either a single TOOL object or a list of TOOL objects.
.tool_choice	Tool-calling behavior: "none", "auto", or "required" (optional).
.json_schema	A JSON schema as an R list to enforce structured output. Passed as response_format = {type: "json_s — enforced at the sampler level.
.grammar	A BNF grammar string to constrain sampling to any formal language (e.g. "root ::= [0-9]{4}" for a 4-digit year). Takes precedence over .json_schema when both are set; use one or the other.
.logprobs	Logical; if TRUE, returns token log-probabilities (default: FALSE).
.top_logprobs	Integer; number of top-alternative log-prob entries to return per token (1–20, optional). Requires .logprobs = TRUE.
.seed	Integer; random seed for reproducible outputs (optional).
.thinking	Logical; if TRUE, enables extended reasoning/thinking mode for models that support it (e.g. Qwen3). NULL (default) lets the server decide; FALSE explicitly disables it for faster responses; TRUE forces it on. The thinking trace is accessible via get_metadata(result)$api_specific$thinking.
.thinking_budget	Integer; maximum tokens the model may use for thinking when .thinking = TRUE. NULL uses the server default (optional).
.server	Base URL of the llama.cpp server. Defaults to the LLAMACPP_SERVER environment variable, falling back to "http://localhost:8080".
.api_key	API key for the llama.cpp server. Defaults to the LLAMACPP_API_KEY environment variable. Leave unset if the server was started without --api-key.
.timeout	Request timeout in seconds (default: 120).
.verbose	If TRUE, displays additional information (default: FALSE).
.dry_run	If TRUE, returns the request object without executing it (default: FALSE).
.max_tries	Maximum retries (default: 3).
.max_tool_rounds	Maximum tool use iterations (default: 10).

Value

A new LLMMessage object containing the original messages plus the assistant's response.

llamacpp_delete_model *Delete a Local GGUF Model File*

Description

Deletes a GGUF file from disk. Issues a warning confirming the deletion. Mirrors the `ollama_delete_model()` pattern.

Usage

```
llamacpp_delete_model(.path)
```

Arguments

`.path`	Full path to the `.gguf` file to delete.

Value

Invisibly returns `TRUE` on success.

llamacpp_download_model *Download a GGUF Model from Hugging Face*

Description

Downloads a single GGUF file from a Hugging Face repository with a streaming progress bar. Use `list_hf_gguf_files()` first to browse available quantizations.

Usage

```
llamacpp_download_model(
  .repo,
  .filename,
  .dir = Sys.getenv("LLAMACPP_MODEL_DIR", "~/models"),
  .timeout = 3600
)
```

Arguments

`.repo`	Hugging Face repository ID (e.g. `"Qwen/Qwen3-8B-GGUF"`).
`.filename`	The exact filename to download (e.g. `"Qwen3-8B-Q4_K_M.gguf"`). Use `list_hf_gguf_files(.repo)` to see what is available.
`.dir`	Destination directory. Defaults to `LLAMACPP_MODEL_DIR` env var, then `"~/models"`. Created if it does not exist.
`.timeout`	Download timeout in seconds (default: 3600).

Value

Invisibly returns the full path of the downloaded file.

`llamacpp_embedding` *Generate Embeddings Using a llama.cpp Server*

Description

Sends text to the `/v1/embeddings` endpoint of a running llama.cpp server and returns embedding vectors.

Usage

```
llamacpp_embedding(
  .input,
  .model = "local-model",
  .server = Sys.getenv("LLAMACPP_SERVER", "http://localhost:8080"),
  .api_key = Sys.getenv("LLAMACPP_API_KEY", ""),
  .timeout = 120,
  .dry_run = FALSE,
  .max_tries = 3
)
```

Arguments

`.input`	A character vector of texts to embed, or an `LLMMessage` object.
`.model`	The model name (default: `"local-model"`). llama.cpp ignores this and serves the loaded embedding model.
`.server`	Base URL of the llama.cpp server. Defaults to the `LLAMACPP_SERVER` environment variable, falling back to `"http://localhost:8080"`.
`.api_key`	API key for the server (default: `LLAMACPP_API_KEY` env var).
`.timeout`	Request timeout in seconds (default: 120).
`.dry_run`	If TRUE, returns the request object without executing it (default: FALSE).
`.max_tries`	Maximum retries (default: 3).

Value

A tibble with columns `input` (text) and `embeddings` (list of numeric vectors).

llamacpp_health *Check Health of the llama.cpp Server*

Description

Calls the /health endpoint of a running llama.cpp server. Returns the status string ("ok", "loading model", "no model loaded", or "error") along with the full parsed response body as a named list.

Usage

```
llamacpp_health(
  .server = Sys.getenv("LLAMACPP_SERVER", "http://localhost:8080"),
  .timeout = 10
)
```

Arguments

.server	Base URL of the llama.cpp server. Defaults to LLAMACPP_SERVER env var, falling back to "http://localhost:8080".
.timeout	Request timeout in seconds (default: 10).

Value

A named list with at least a status element.

llamacpp_list_local_models *List Local GGUF Model Files*

Description

Scans a directory for .gguf files and returns a tibble with their names, sizes, and modification times. No server needed.

Usage

```
llamacpp_list_local_models(
  .path = Sys.getenv("LLAMACPP_MODEL_DIR", "~/models")
)
```

Arguments

.path	Directory to scan (default: LLAMACPP_MODEL_DIR env var, then "~/models"). The path is expanded with path.expand().

Value

A tibble with columns filename, size_gb, modified, and path.

llamacpp_list_models *List Models Loaded in the llama.cpp Server*

Description

Calls the `/v1/models` endpoint of a running llama.cpp server and returns the currently loaded model(s) as a tibble. In normal operation this is one row; two rows appear when speculative decoding is active (main + draft model).

Usage

```
llamacpp_list_models(
  .server = Sys.getenv("LLAMACPP_SERVER", "http://localhost:8080"),
  .api_key = Sys.getenv("LLAMACPP_API_KEY", ""),
  .timeout = 30,
  .max_tries = 3
)
```

Arguments

`.server`	Base URL of the llama.cpp server. Defaults to `LLAMACPP_SERVER` env var, falling back to `"http://localhost:8080"`.
`.api_key`	API key for the server (default: `LLAMACPP_API_KEY` env var).
`.timeout`	Request timeout in seconds (default: 30).
`.max_tries`	Maximum retries (default: 3).

Value

A tibble with columns `id`, `object`, and `created`.

llamacpp_rerank *Rerank Documents Using a llama.cpp Server*

Description

Calls the `/v1/reranking` endpoint of a running llama.cpp server to score documents by relevance to a query. Useful for building fully-local RAG pipelines (embed → cosine search → rerank → chat, all with `llamacpp()`).

Usage

```
llamacpp_rerank(
  .query,
  .documents,
  .model = "local-model",
  .server = Sys.getenv("LLAMACPP_SERVER", "http://localhost:8080"),
  .api_key = Sys.getenv("LLAMACPP_API_KEY", ""),
  .timeout = 60,
  .max_tries = 3,
  .dry_run = FALSE
)
```

Arguments

.query	A single query string.
.documents	A character vector of documents to rerank.
.model	The model name (default: "local-model").
.server	Base URL of the llama.cpp server. Defaults to LLAMACPP_SERVER env var, falling back to "http://localhost:8080".
.api_key	API key for the server (default: LLAMACPP_API_KEY env var).
.timeout	Request timeout in seconds (default: 60).
.max_tries	Maximum retries (default: 3).
.dry_run	If TRUE, returns the request object without executing it (default: FALSE).

Value

A tibble with columns index (original position), document (text), and relevance_score, sorted by descending score.

LLMMessage *Large Language Model Message Class*

Description

LLMMessage is an S7 class for managing a conversation history intended for use with large language models (LLMs). Please use llm_message()to create or modify LLMMessage objects.

Usage

```
LLMMessage(message_history = list(), system_prompt = character(0))
```

Arguments

message_history	A list containing messages. Each message is a named list with keys like role, content, media, etc.
system_prompt	A character string representing the default system prompt used for the conversation.

Details

The LLMMessage class includes the following features:

- Stores message history in a structured format.
- Supports attaching media and metadata to messages.
- Provides generics like add_message(), has_image(), and remove_message() for interaction.
- Enables API-specific formatting through the to_api_format() generic.
- message_history: A list containing messages. Each message is a named list with keys like role, content, media, etc.
- system_prompt: A character string representing the default system prompt used for the conversation.

---

llm_message *Create or Update Large Language Model Message Object*

Description

This function creates a new LLMMessage object or updates an existing one. It supports adding text prompts and various media types, such as images, PDFs, text files, or plots.

Usage

```
llm_message(
  .llm = NULL,
  .prompt = NULL,
  .role = "user",
  .system_prompt = "You are a helpful assistant",
  .media = NULL,
  .files = NULL,
  .imagefile = NULL,
  .pdf = NULL,
  .textfile = NULL,
  .capture_plot = FALSE,
  .f = NULL
)
```

Arguments

.llm	An existing LLMMessage object or an initial text prompt.
.prompt	Text prompt to add to the message history.
.role	The role of the message sender, typically "user" or "assistant".
.system_prompt	Default system prompt if a new LLMMessage needs to be created.
.media	An inline media object or a list of them. Accepted types: img(), audio_file(), video_file(), pdf_file().
.files	A tidyllm_file object or list of them returned by upload_file(). These are remote file references stored on a provider's server.
.imagefile	Path to an image file to be attached (optional). Deprecated; use .media = img(path) instead.
.pdf	Path to a PDF file to be attached (optional). Deprecated; use .media = pdf_file(path) instead.
.textfile	Path to a text file to be read and attached (optional).
.capture_plot	Boolean to indicate whether the current plot should be captured and attached as an image, the same way as .media = img(path) (optional).
.f	An R function or an object coercible to a function via rlang::as_function, whose output should be captured and attached (optional).

Value

Returns an updated or new LLMMessage object.

See Also

df_llm_message()

Other Message Creation Utilities: df_llm_message()

---

mistral *Mistral Provider Function*

Description

The mistral() function acts as an interface for interacting with the Mistral API through main tidyllm verbs such as chat() and embed(). It dynamically routes requests to Mistral-specific functions like mistral_chat() and mistral_embedding() based on the context of the call.

Usage

```
mistral(..., .called_from = NULL)
```

Arguments

...	Parameters to be passed to the appropriate Mistral-specific function, such as model configuration, input text, or API-specific options.
.called_from	An internal argument that specifies which action (e.g., chat, embed, send_batch) the function is being invoked from. This argument is automatically managed and should not be modified by the user.

Value

The result of the requested action, depending on the specific function invoked (e.g., an updated LLMMessage object for chat(), or a matrix for embed()).

---

mistral_chat *Send LLMMessage to Mistral API*

Description

Send LLMMessage to Mistral API

Usage

```
mistral_chat(
  .llm,
  .model = "zai-glm-5-3",
  .frequency_penalty = NULL,
  .logit_bias = NULL,
  .presence_penalty = NULL,
  .seed = NULL,
  .stop = NULL,
  .stream = FALSE,
  .temperature = 0.7,
  .top_p = 1,
  .min_tokens = NULL,
  .max_tokens = NULL,
  .json_schema = NULL,
  .safe_prompt = FALSE,
  .reasoning_effort = NULL,
  .timeout = 120,
  .max_tries = 3,
  .dry_run = FALSE,
  .verbose = FALSE,
  .tools = NULL,
  .tool_choice = NULL,
  .max_tool_rounds = 10
)
```

Arguments

.llm	An LLMMessage object.
.model	The model identifier to use (default: "zai-glm-5-3").
.frequency_penalty	Number between -2.0 and 2.0. Positive values penalize new tokens based on their existing frequency.
.logit_bias	A named list modifying the likelihood of specified tokens appearing in the completion.
.presence_penalty	Number between -2.0 and 2.0. Positive values penalize new tokens based on whether they appear in the text so far.
.seed	If specified, the system will make a best effort to sample deterministically.
.stop	Up to 4 sequences where the API will stop generating further tokens.
.stream	If set to TRUE, the answer will be streamed to console as it comes (default: FALSE).
.temperature	What sampling temperature to use, between 0 and 2. Higher values make the output more random.
.top_p	An alternative to sampling with temperature, called nucleus sampling.
.min_tokens	The minimum number of tokens to generate in the completion. Must be >= 0 (optional).
.max_tokens	An upper bound for the number of tokens that can be generated for a completion.
.json_schema	A JSON schema object provided by tidyllm schema or ellmer schemata.
.safe_prompt	Whether to inject a safety prompt before all conversations (default: FALSE).
.reasoning_effort	Controls how much reasoning models think before answering; one of "none", "low", "medium", or "high" (default: NULL, meaning the API default). Supported values vary by model: mistral-large-4 accepts only "high" and "none", and "none" turns reasoning off.
.timeout	When should our connection time out in seconds (default: 120).
.max_tries	Maximum retries to peform request
.dry_run	If TRUE, perform a dry run and return the request object (default: FALSE).
.verbose	Should additional information be shown after the API call? (default: FALSE)
.tools	Either a single TOOL object or a list of TOOL objects representing the available functions for tool calls.
.tool_choice	A character string specifying the tool-calling behavior; valid values are "none", "auto", or "required".
.max_tool_rounds	Integer specifying the maximum number of tool use iterations (default: 10). Set to 1 for single-round tool use, or higher for multi-turn agentic loops.

Value

Returns an updated LLMMessage object.

mistral_embedding *Generate Embeddings Using Mistral API*

Description

Generate Embeddings Using Mistral API

Usage

```
mistral_embedding(
  .input,
  .model = "mistral-embed",
  .timeout = 120,
  .max_tries = 3,
  .dry_run = FALSE
)
```

Arguments

.input	A character vector of texts to embed or an LLMMessage object
.model	The embedding model identifier (default: "mistral-embed").
.timeout	Timeout for the API request in seconds (default: 120).
.max_tries	Maximum retries to peform request
.dry_run	If TRUE, perform a dry run and return the request object.

Value

A matrix where each column corresponds to the embedding of a message in the message history.

mistral_list_models *List Available Models from the Mistral API*

Description

List Available Models from the Mistral API

Usage

```
mistral_list_models(
  .api_url = "https://api.mistral.ai",
  .timeout = 60,
  .max_tries = 3,
  .dry_run = FALSE,
  .verbose = FALSE
)
```

Arguments

.api_url	Base URL for the API (default: "https://api.mistral.ai").
.timeout	Request timeout in seconds (default: 60).
.max_tries	Maximum number of retries for the API request (default: 3).
.dry_run	Logical; if TRUE, returns the prepared request object without executing it.
.verbose	Logical; if TRUE, prints additional information about the request.

Value

A tibble containing model information (columns include id and created), or NULL if no models are found.

---

ollama *Ollama API Provider Function*

Description

The ollama() function acts as an interface for interacting with local AI models via the Ollama API. It integrates seamlessly with the main tidyllm verbs such as chat() and embed().

Usage

```
ollama(..., .called_from = NULL)
```

Arguments

...	Parameters to be passed to the appropriate Ollama-specific function, such as model configuration, input text, or API-specific options.
.called_from	An internal argument specifying the verb (e.g., chat, embed) the function is invoked from. This argument is automatically managed by tidyllm and should not be set by the user.

Details

Some functionalities, like ollama_download_model() or ollama_list_models() are unique to the Ollama API and do not have a general verb counterpart. These functions can be only accessed directly.

Supported Verbs:

- chat(): Sends a message to an Ollama model and retrieves the model's response.
- embed(): Generates embeddings for input texts using an Ollama model.
- send_batch(): Behaves different than the other send_batch() verbs since it immediately processes the answers

Value

The result of the requested action:

- For `chat()`: An updated `LLMMessage` object containing the model's response.
- For `embed()`: A matrix where each column corresponds to an embedding.

`ollama_chat` *Interact with local AI models via the Ollama API*

Description

Interact with local AI models via the Ollama API

Usage

```
ollama_chat(
  .llm,
  .model = "qwen3.5:4b",
  .stream = FALSE,
  .seed = NULL,
  .json_schema = NULL,
  .temperature = NULL,
  .num_ctx = 32768,
  .num_predict = NULL,
  .top_k = NULL,
  .top_p = NULL,
  .min_p = NULL,
  .mirostat = NULL,
  .mirostat_eta = NULL,
  .mirostat_tau = NULL,
  .repeat_last_n = NULL,
  .repeat_penalty = NULL,
  .tools = NULL,
  .max_tool_rounds = 10,
  .tfs_z = NULL,
  .stop = NULL,
  .think = NULL,
  .ollama_server = "http://localhost:11434",
  .timeout = 120,
  .max_tries = 3,
  .keep_alive = NULL,
  .dry_run = FALSE
)
```

Arguments

.llm	An LLMMessage object containing the conversation history and system prompt.
.model	Character string specifying the Ollama model to use (default: "qwen3-vl")
.stream	Logical; whether to stream the response (default: FALSE)
.seed	Integer; seed for reproducible generation (default: NULL)
.json_schema	A JSON schema object as R list to enforce the output structure (default: NULL)
.temperature	Float between 0-2; controls randomness in responses (default: NULL)
.num_ctx	Integer; sets the context window size (default: 32768)
.num_predict	Integer; maximum number of tokens to predict (default: NULL)
.top_k	Integer; controls diversity by limiting top tokens considered (default: NULL)
.top_p	Float between 0-1; nucleus sampling threshold (default: NULL)
.min_p	Float between 0-1; minimum probability threshold (default: NULL)
.mirostat	Integer (0,1,2); enables Mirostat sampling algorithm (default: NULL)
.mirostat_eta	Float; Mirostat learning rate (default: NULL)
.mirostat_tau	Float; Mirostat target entropy (default: NULL)
.repeat_last_n	Integer; tokens to look back for repetition (default: NULL)
.repeat_penalty	Float; penalty for repeated tokens (default: NULL)
.tools	Either a single TOOL object or a list of TOOL objects representing the available functions for tool calls.
.max_tool_rounds	Integer; maximum number of tool use iterations for multi-turn tool calling (default: 10). Set to 1 for single-round tool use, or higher for multi-turn agentic loops.
.tfs_z	Float; tail free sampling parameter (default: NULL)
.stop	Character; custom stop sequence(s) (default: NULL)
.think	Logical or character; controls thinking mode for supported models like Qwen3. Use FALSE to disable, TRUE to enable, or "high"/"medium"/"low" to set effort level (default: NULL - model default)
.ollama_server	String; Ollama API endpoint (default: "http://localhost:11434")
.timeout	Integer; API request timeout in seconds (default: 120)
.max_tries	Integer; maximum number of retries for the request (default: 3)
.keep_alive	Character; How long should the ollama model be kept in memory after request (default: NULL - 5 Minutes)
.dry_run	Logical; if TRUE, returns request object without execution (default: FALSE)

Details

The function provides extensive control over the generation process through various parameters:

- Temperature (0-2): Higher values increase creativity, lower values make responses more focused
- Top-k/Top-p: Control diversity of generated text
- Mirostat: Advanced sampling algorithm for maintaining consistent complexity
- Repeat penalties: Prevent repetitive text
- Context window: Control how much previous conversation is considered

Value

A new LLMMessage object containing the original messages plus the model's response

Examples

```
## Not run:
llm_message("user", "Hello, how are you?")
response <- ollama_chat(llm, .model = "gemma2", .temperature = 0.7)

# With custom parameters
response <- ollama_chat(
  llm,
  .model = "llama2",
  .temperature = 0.8,
  .top_p = 0.9,
  .num_ctx = 4096
)

## End(Not run)
```

---

ollama_delete_model *Delete a model from the Ollama API*

Description

This function sends a DELETE request to the Ollama API to remove a specified model.

Usage

```
ollama_delete_model(.model, .ollama_server = "http://localhost:11434")
```

Arguments

.model	The name of the model to delete.
.ollama_server	The base URL of the Ollama API (default is "http://localhost:11434").

ollama_download_model *Download a model from the Ollama API*

Description

This function sends a request to the Ollama API to download a specified model from Ollama's large online library of models.

Usage

```
ollama_download_model(.model, .ollama_server = "http://localhost:11434")
```

Arguments

.model	The name of the model to download.
.ollama_server	The base URL of the Ollama API (default is "http://localhost:11434").

ollama_embedding *Generate Embeddings Using Ollama API*

Description

Generate Embeddings Using Ollama API

Usage

```
ollama_embedding(
  .input,
  .model = "qwen3-embedding:0.6b",
  .truncate = TRUE,
  .ollama_server = "http://localhost:11434",
  .timeout = 120,
  .dry_run = FALSE
)
```

Arguments

.input	Aa charachter vector of texts to embed or an LLMMessage object
.model	The embedding model identifier (default: "all-minilm").
.truncate	Whether to truncate inputs to fit the model's context length (default: TRUE).
.ollama_server	The URL of the Ollama server to be used (default: "http://localhost:11434").
.timeout	Timeout for the API request in seconds (default: 120).
.dry_run	If TRUE, perform a dry run and return the request object.

Value

A matrix where each column corresponds to the embedding of a message in the message history.

---

ollama_list_models *Retrieve and return model information from the Ollama API*

Description

This function connects to the Ollama API and retrieves information about available models, returning it as a tibble.

Usage

```
ollama_list_models(.ollama_server = "http://localhost:11434")
```

Arguments

`.ollama_server`	The URL of the ollama server to be used

Value

A tibble containing model information, or NULL if no models are found.

---

openai *The* `openai()` *function acts as an interface for interacting with the OpenAI API through main* `tidyllm` *verbs such as* `chat()`*,* `embed()`*, and* `send_batch()`*. Chat uses the Responses API (*`POST /v1/responses`*); embeddings and batch operations use the Chat Completions / Embeddings endpoints unchanged.*

Description

The `openai()` function acts as an interface for interacting with the OpenAI API through main `tidyllm` verbs such as `chat()`, `embed()`, and `send_batch()`. Chat uses the Responses API (`POST /v1/responses`); embeddings and batch operations use the Chat Completions / Embeddings endpoints unchanged.

Usage

```
openai(..., .called_from = NULL)
```

Arguments

`...`	Parameters passed to the appropriate OpenAI-specific function.
`.called_from`	Internal routing argument; do not set manually.

Value

Result of the requested action.

`openai_chat` *Send LLM Messages to the OpenAI Responses API*

Description

Sends a message history to the OpenAI Responses API (`POST /v1/responses`) and returns the assistant's reply. Supports streaming, tool use, structured output, and reasoning models (o-series) via `.reasoning_effort`.

Usage

```
openai_chat(
  .llm,
  .model = "gpt-6-luna",
  .max_output_tokens = NULL,
  .temperature = NULL,
  .seed = NULL,
  .stream = FALSE,
  .timeout = 60,
  .verbose = FALSE,
  .json_schema = NULL,
  .max_tries = 3,
  .dry_run = FALSE,
  .reasoning_effort = NULL,
  .tools = NULL,
  .tool_choice = NULL,
  .max_tool_rounds = 10,
  .stateful = FALSE
)
```

Arguments

`.llm`	An `LLMMessage` object containing the conversation history.
`.model`	The model identifier (default: `"gpt-6-luna"`).
`.max_output_tokens`	Maximum tokens to generate (caps reasoning + completion).
`.temperature`	Sampling temperature (0-2).
`.seed`	Seed for deterministic sampling.
`.stream`	If TRUE, stream output to console (default: FALSE).
`.timeout`	Request timeout in seconds (default: 60).
`.verbose`	Print rate-limit info after the call (default: FALSE).

.json_schema	A tidyllm schema or ellmer type for structured output.
.max_tries	Maximum retry attempts (default: 3).
.dry_run	If TRUE, return the request object without sending (default: FALSE).
.reasoning_effort	For o-series models: "low", "medium", or "high".
.tools	A TOOL object or list of TOOL objects for function calling.
.tool_choice	Tool selection behavior: "none", "auto", or "required".
.max_tool_rounds	Maximum tool-loop iterations (default: 10).
.stateful	If TRUE, use previous_response_id to pass conversation context server-side instead of re-sending full message history. Falls back to full-history mode silently when no previous response ID is available (first turn or prior turn used a different provider), and with a warning if the server rejects the stored ID (e.g., expired context). Default: FALSE.

Value

A new LLMMessage object with the assistant's response appended.

---

openai_check_research *Check the Status of an OpenAI Background Research Job*

Description

Polls the status of an OpenAI background response created by openai_deep_research(.background = TRUE).

Usage

```
openai_check_research(.job, .max_tries = 3)
```

Arguments

.job	A tidyllm_research_job object returned by openai_deep_research(.background = TRUE).
.max_tries	Maximum retries per HTTP request (default: 3).

Value

An updated tidyllm_research_job with $status set. If completed, $response is also populated and ready for fetch_job().

openai_code_interpreter *OpenAI built-in code interpreter tool (server-executed)*

Description

OpenAI built-in code interpreter tool (server-executed)

Usage

```
openai_code_interpreter()
```

Value

A TOOL object declaring the code_interpreter built-in.

openai_deep_research *Submit a Deep Research Request to OpenAI*

Description

Sends a research request to OpenAI via the Responses API with `background: true` and the web search tool. The model autonomously searches the web and synthesises a long-form answer, which can take 5-30 minutes. OpenAI has retired its dedicated deep research models (`o3-deep-research`, `o4-mini-deep-research`); a GPT-6 model with web search does the job now.

Usage

```
openai_deep_research(
  .llm,
  .model = "gpt-6-sol",
  .background = FALSE,
  .reasoning_effort = "medium",
  .json_schema = NULL,
  .max_output_tokens = NULL,
  .timeout = 1800,
  .max_tries = 3
)
```

Arguments

`.llm`	An `LLMMessage` object containing the research question.
`.model`	The model to use (default: `"gpt-6-sol"`).
`.background`	Logical; if `TRUE`, returns a `tidyllm_research_job` immediately without waiting for completion (default: `FALSE`).

.reasoning_effort
: Reasoning level for the model: "low", "medium" (default), or "high".

.json_schema
: A tidyllm schema for structured JSON output (optional).

.max_output_tokens
: Maximum tokens to generate (default: NULL for model default).

.timeout
: Seconds to wait in blocking mode before giving up (default: 1800).

.max_tries
: Maximum retries per HTTP request (default: 3).

Value

If .background = FALSE, an updated LLMMessage with the research reply. If .background = TRUE, a tidyllm_research_job for use with check_job()/fetch_job().

---

openai_embedding *Generate Embeddings Using OpenAI API*

Description

Generate Embeddings Using OpenAI API

Usage

```
openai_embedding(
  .input,
  .model = "text-embedding-3-small",
  .truncate = TRUE,
  .timeout = 120,
  .dry_run = FALSE,
  .max_tries = 3,
  .verbose = FALSE
)
```

Arguments

.input
: An existing LLMMessage object (or a character vector of texts to embed)

.model
: The embedding model identifier (default: "text-embedding-3-small").

.truncate
: Whether to truncate inputs to fit the model's context length (default: TRUE).

.timeout
: Timeout for the API request in seconds (default: 120).

.dry_run
: If TRUE, perform a dry run and return the request object.

.max_tries
: Maximum retry attempts for requests (default: 3).

.verbose
: Should information about current ratelimits be printed? (default: FALSE)

Value

A tibble with two columns: input and embeddings. The input column contains the texts sent to embed, and the embeddings column is a list column where each row contains an embedding vector of the sent input.

openai_fetch_research *Fetch Results from a Completed OpenAI Deep Research Job*

Description

Extracts the assistant reply from a completed `tidyllm_research_job` returned by `openai_deep_research(.background = TRUE)`.

Usage

```
openai_fetch_research(.job, .max_tries = 3)
```

Arguments

`.job`	A `tidyllm_research_job` object. If not yet completed, polls once via `openai_check_research()` and errors if still incomplete.
`.max_tries`	Maximum retries per HTTP request (default: 3).

Value

An updated `LLMMessage` with the research reply appended.

openai_list_models *List Available Models from the OpenAI API*

Description

List Available Models from the OpenAI API

Usage

```
openai_list_models(
  .api_url = "https://api.openai.com",
  .timeout = 60,
  .max_tries = 3,
  .dry_run = FALSE,
  .verbose = FALSE
)
```

Arguments

`.api_url`	Base URL for the API (default: "https://api.openai.com").
`.timeout`	Request timeout in seconds (default: 60).
`.max_tries`	Maximum number of retries for the API request (default: 3).
`.dry_run`	Logical; if TRUE, returns the prepared request object without executing it.
`.verbose`	Logical; if TRUE, prints additional information about the request.

Value

A tibble containing model information (columns include `id`, `created`, and `owned_by`), or NULL if no models are found.

`openai_websearch` *OpenAI built-in web search tool (server-executed)*

Description

OpenAI built-in web search tool (server-executed)

Usage

```
openai_websearch()
```

Value

A TOOL object declaring the web_search_preview built-in.

`openrouter` *OpenRouter Provider Function*

Description

The `openrouter()` function provides access to hundreds of AI models from different providers through the OpenRouter API, using a single OpenAI-compatible interface.

Usage

```
openrouter(..., .called_from = NULL)
```

Arguments

`...`	Parameters passed to the appropriate OpenRouter-specific function.
`.called_from`	An internal argument specifying which verb invoked this function. Managed automatically by tidyllm verbs; do not set manually.

Value

The result of the requested action (e.g., an updated `LLMMessage` for `chat()`).

`openrouter_chat` *Send LLM Messages to the OpenRouter Chat API*

Description

Sends a message history to the OpenRouter API, which provides access to hundreds of models from different providers through a single OpenAI-compatible endpoint.

Usage

```
openrouter_chat(
  .llm,
  .model = "anthropic/claude-sonnet-5.5",
  .max_tokens = 2048,
  .temperature = NULL,
  .top_p = NULL,
  .frequency_penalty = NULL,
  .presence_penalty = NULL,
  .stop = NULL,
  .stream = FALSE,
  .json_schema = NULL,
  .tools = NULL,
  .tool_choice = NULL,
  .reasoning = NULL,
  .provider = NULL,
  .route = NULL,
  .models = NULL,
  .api_url = "https://openrouter.ai",
  .timeout = 60,
  .verbose = FALSE,
  .dry_run = FALSE,
  .max_tries = 3,
  .max_tool_rounds = 10
)
```

Arguments

`.llm`	An `LLMMessage` object containing the conversation history.
`.model`	The model identifier to use (default: `"google/gemini-3.6-flash"`). Any model available on OpenRouter can be used.
`.max_tokens`	Maximum number of tokens in the response (default: 2048).
`.temperature`	Controls randomness (0–2, optional).
`.top_p`	Nucleus sampling parameter (0–1, optional).
`.frequency_penalty`	Penalizes repeated tokens (-2 to 2, optional).

`.presence_penalty`
: Encourages new topics (-2 to 2, optional).

`.stop`
: One or more stop sequences (optional).

`.stream`
: Logical; if TRUE, streams the response (default: FALSE).

`.json_schema`
: A JSON schema object for structured output (default: NULL).

`.tools`
: Either a single TOOL object or a list of TOOL objects for tool calls.

`.tool_choice`
: Tool-calling behavior: `"none"`, `"auto"`, or `"required"` (optional).

`.reasoning`
: A named list controlling reasoning token behavior (optional). Supported fields vary by model family:

  - `effort`: one of `"xhigh"`, `"high"`, `"medium"`, `"low"`, `"minimal"`, `"none"` (OpenAI/Grok)
  - `max_tokens`: integer specifying the reasoning token budget (Anthropic/Gemini/Alibaba)
  - `exclude`: logical; if `TRUE`, reasoning is used internally but not returned in the response
  - `enabled`: logical; if `TRUE`, activates reasoning at default settings

  Example: `list(effort = "high")` or `list(max_tokens = 4000, exclude = FALSE)`.

`.provider`
: A named list of OpenRouter provider preferences, e.g. `list(order = c("Anthropic", "AWS Bedrock"), allow_fallbacks = TRUE)` (optional).

`.route`
: OpenRouter routing strategy; `"fallback"` routes to the next model if the primary is unavailable (optional).

`.models`
: A character vector of model IDs to use as fallbacks when the primary model is unavailable (optional). Used together with `.route = "fallback"`.

`.api_url`
: Base URL for the OpenRouter API (default: `"https://openrouter.ai"`).

`.timeout`
: Request timeout in seconds (default: 60).

`.verbose`
: If TRUE, displays additional information after the API call (default: FALSE).

`.dry_run`
: If TRUE, returns the request object without executing it (default: FALSE).

`.max_tries`
: Maximum retries (default: 3).

`.max_tool_rounds`
: Maximum number of tool use iterations (default: 10).

Value

A new `LLMMessage` object containing the original messages plus the assistant's response.

openrouter_credits *Get OpenRouter Credit Balance*

Description

Returns the total credits purchased and total usage so far for the current API key.

Usage

```
openrouter_credits(
  .api_url = "https://openrouter.ai",
  .timeout = 60,
  .max_tries = 3
)
```

Arguments

.api_url	Base URL for the OpenRouter API (default: "https://openrouter.ai").
.timeout	Request timeout in seconds (default: 60).
.max_tries	Maximum retries (default: 3).

Value

A tibble with columns total_credits (USD purchased), total_usage (USD consumed so far), and remaining (USD remaining).

openrouter_embedding *Generate Embeddings Using the OpenRouter API*

Description

Sends text to an embedding model accessible via OpenRouter and returns embedding vectors. Note that embedding models are not listed in list_models(openrouter()) — specify the model ID directly. Known supported models include "openai/text-embedding-3-small", "openai/text-embedding-3-large", and "mistralai/mistral-embed".

Usage

```
openrouter_embedding(
  .input,
  .model = "openai/text-embedding-3-small",
  .timeout = 120,
  .dry_run = FALSE,
  .max_tries = 3
)
```

Arguments

.input	An LLMMessage object or a character vector of texts to embed.
.model	The embedding model ID (default: "openai/text-embedding-3-small").
.timeout	Request timeout in seconds (default: 120).
.dry_run	If TRUE, returns the request object without executing it (default: FALSE).
.max_tries	Maximum retries (default: 3).

Value

A tibble with columns input (text) and embeddings (list of numeric vectors).

---

openrouter_generation *Get Details for an OpenRouter Generation*

Description

Fetches cost, latency, and token details for a past generation using its ID. The generation ID is available in get_metadata(result)$api_specific$id after a chat() call.

Usage

```
openrouter_generation(
  .id,
  .api_url = "https://openrouter.ai",
  .timeout = 60,
  .max_tries = 3
)
```

Arguments

.id	The generation ID string (e.g. "gen-..."") returned by OpenRouter.
.api_url	Base URL for the OpenRouter API (default: "https://openrouter.ai").
.timeout	Request timeout in seconds (default: 60).
.max_tries	Maximum retries (default: 3).

Value

A named list with generation details including id, model, total_cost, tokens_prompt, tokens_completion, and latency.

openrouter_list_models *List Available Models on OpenRouter*

Description

Retrieves the list of models available through the OpenRouter API, including pricing and context window information.

Usage

```
openrouter_list_models(
  .api_url = "https://openrouter.ai",
  .timeout = 60,
  .max_tries = 3,
  .dry_run = FALSE,
  .verbose = FALSE
)
```

Arguments

.api_url	Base URL for the OpenRouter API (default: "https://openrouter.ai").
.timeout	Request timeout in seconds (default: 60).
.max_tries	Maximum retries (default: 3).
.dry_run	If TRUE, returns the request object without executing it (default: FALSE).
.verbose	If TRUE, displays additional information (default: FALSE).

Value

A tibble with columns id, name, context_length, prompt_price_per_million, and completion_price_per_million, or NULL if no models are found.

parallel_chat *Send many chats at once*

Description

Performs a list of LLMMessages against one provider concurrently and returns their replies in the same order, with the same names. It is the middle ground between chat() in a loop, which waits for each answer before starting the next, and send_batch(), which is cheaper still but returns hours later.

Usage

```
parallel_chat(
  .llms,
  .provider = getOption("tidyllm_chat_default"),
  .max_active = 4,
  .throttle = NULL,
  .on_error = "continue",
  .temperature = NULL,
  .timeout = NULL,
  .top_p = NULL,
  .max_tries = NULL,
  .model = NULL,
  .verbose = NULL,
  .json_schema = NULL,
  .seed = NULL,
  .stop = NULL,
  .frequency_penalty = NULL,
  .presence_penalty = NULL,
  .stream = FALSE,
  .tools = NULL
)
```

Arguments

.llms	A list of LLMMessage objects. Names are preserved.
.provider	A provider function call, as in chat().
.max_active	Maximum number of requests in flight at once. Keep it modest against a rate-limited provider: httr2 applies retries across the whole set rather than per request, so a high number is a good way to collect 429s.
.throttle	Optional maximum number of requests per second, applied across the set. The straightforward defence against a rate limit.
.on_error	"continue" (default) keeps going and puts the failure in that element's slot; "stop" aborts the whole set on the first failure.
.temperature	Numeric; controls the randomness of the model's output (0 = deterministic).
.timeout	Numeric; the maximum time (in seconds) to wait for a response.
.top_p	Numeric; nucleus sampling parameter, which limits the sampling to the top cumulative probability p.
.max_tries	Integer; the maximum number of retries for failed requests.
.model	Character; the model identifier to use (e.g., "gpt-4").
.verbose	Logical; if TRUE, prints additional information about the request and response.
.json_schema	List; A JSON schema object as R list to enforce the output structure
.seed	Integer; sets a random seed for reproducibility.
.stop	Character vector; specifies sequences where the model should stop generating further tokens.

`.frequency_penalty`
: Numeric; adjusts the likelihood of repeating tokens (positive values decrease repetition).

`.presence_penalty`
: Numeric; adjusts the likelihood of introducing new tokens (positive values encourage novelty).

`.stream`
: Logical; if `TRUE`, streams the response from the provider in real-time.

`.tools`
: Either a single TOOL object or a list of TOOL objects representing the available functions for tool calls.

Details

Neither streaming nor tool calls are supported here, and both are refused rather than quietly ignored. A tool call is a conversation, not a request: its rounds would run one after another after the parallel phase, which is a surprising performance cliff rather than a feature. Use `send_chat()` for those, which can have several conversations in flight at once.

Value

A list the same length as `.llms`. Successful elements are `LLMMessage` objects; failed ones, under `.on_error = "continue"`, are the condition that failed, so nothing is silently dropped and a downstream `get_reply()` fails loudly on exactly the elements that have no reply.

Examples

```
## Not run:
questions <- list(
  physics = llm_message("What is a photon?"),
  biology = llm_message("What is a ribosome?")
)
answers <- parallel_chat(questions, claude())
purrr::map_chr(answers, get_reply)

## End(Not run)
```

---

pdf_file *Create a PDF Object*

Description

Stores a reference to a local PDF file along with a pdftools text extraction. Binary encoding is deferred to format time; providers that support binary PDF encode on demand, others use the stored text fallback.

Usage

```
pdf_file(.path, pages = NULL, .text_extract = FALSE, engine = NULL)
```

Arguments

.path	The path to the PDF file on disk.
pages	An integer vector of page numbers to include (e.g. 1:3). NULL uses all pages.
.text_extract	Logical; if TRUE always use pdftools text extraction regardless of provider (equivalent to the old .pdf parameter behaviour). Default FALSE uses binary-first.
engine	Optional engine name (e.g. "mistral-ocr"); a warning is issued if the provider does not support binary PDF and the engine choice cannot be honoured.

Value

A tidyllm_pdf object.

---

pdf_page_batch *Batch Process PDF into LLM Messages*

Description

This function processes a PDF file page by page. For each page, it extracts the text and converts the page into an image. It creates a list of LLMMessage objects with the text and the image for multimodal processing. Users can specify a range of pages to process and provide a custom function to generate prompts for each page.

Usage

```
pdf_page_batch(
  .pdf,
  .general_prompt,
  .system_prompt = "You are a helpful assistant",
  .page_range = NULL,
  .prompt_fn = NULL
)
```

Arguments

.pdf	Path to the PDF file.
.general_prompt	A default prompt that is applied to each page if .prompt_fn is not provided.
.system_prompt	Optional system prompt to initialize the LLMMessage (default is "You are a helpful assistant").
.page_range	A vector of two integers specifying the start and end pages to process. If NULL, all pages are processed.
.prompt_fn	An optional custom function that generates a prompt for each page. The function takes the page text as input and returns a string. If NULL, .general_prompt is used for all pages.

Value

A named list of LLMMessage objects, each containing the text and image for a page. Names are `page_<n>`, with `<n>` the page number in the original document, so they survive `parallel_chat()` and identify rows in its output.

`perplexity` *Perplexity Provider Function*

Description

The `perplexity()` function acts as a provider interface for interacting with the Perplexity API through `tidyllm`'s `chat()` verb. It dynamically routes requests to Perplxeity-specific function. At the moment this is only `perplexity_chat()`

Usage

```
perplexity(..., .called_from = NULL)
```

Arguments

`...`	Parameters to be passed to the appropriate Perplexity-specific function, such as model configuration, input text, or API-specific options.
`.called_from`	An internal argument specifying which action (e.g., `chat`, `embed`) the function is invoked from. This argument is automatically managed by the `tidyllm` verbs and should not be modified by the user.

Value

The result of the requested action, depending on the specific function invoked (e.g., an updated `LLMMessage` object for `chat()`).

Deprecated

Perplexity support is deprecated since tidyllm 0.7.0 and ends in 0.8.0. Use `openrouter(.model = "perplexity/sonar")` for the same models, or `websearch_tool()` to give any provider web search.

perplexity_chat *Send LLM Messages to the Perplexity Chat API*

Description

Sends a chat message history to the Perplexity Chat API, supporting all documented API parameters.

Usage

```
perplexity_chat(
  .llm,
  .model = "sonar",
  .max_tokens = 1024,
  .temperature = NULL,
  .top_p = NULL,
  .frequency_penalty = NULL,
  .presence_penalty = NULL,
  .stop = NULL,
  .search_domain_filter = NULL,
  .search_language_filter = NULL,
  .language_preference = NULL,
  .return_images = FALSE,
  .image_domain_filter = NULL,
  .image_format_filter = NULL,
  .search_recency_filter = NULL,
  .search_mode = "web",
  .search_after_date_filter = NULL,
  .search_before_date_filter = NULL,
  .last_updated_after_filter = NULL,
  .last_updated_before_filter = NULL,
  .disable_search = FALSE,
  .enable_search_classifier = FALSE,
  .reasoning_effort = NULL,
  .return_related_questions = FALSE,
  .user_location = NULL,
  .search_context_size = NULL,
  .search_type = NULL,
  .stream_mode = NULL,
  .json_schema = NULL,
  .top_k = NULL,
  .web_search_options = NULL,
  .api_url = "https://api.perplexity.ai/",
  .timeout = 60,
  .stream = FALSE,
  .verbose = FALSE,
  .max_tries = 3,
```

```
  .dry_run = FALSE
)
```

Arguments

.llm	An LLMMessage object containing the conversation history.
.model	Model name to use (default: "sonar").
.max_tokens	Max completion tokens (default: 1024).
.temperature	Controls response randomness (0 < x < 2).
.top_p	Nucleus sampling threshold (0 < x < 1).
.frequency_penalty	Number > 0. Penalizes frequent tokens.
.presence_penalty	Numeric between -2 and 2. Penalizes present tokens.
.stop	Stop sequence(s), string or character vector/list.
.search_domain_filter	Character vector of domains to allowlist/denylist (max 10; prefix with "-" to denylist).
.search_language_filter	ISO 639-1 language code to restrict search results (e.g. "en", "de", "fr").
.language_preference	ISO 639-1 code for preferred response language (e.g. "en").
.return_images	Logical; if TRUE, returns images from search.
.image_domain_filter	Character vector of domains to restrict image results to.
.image_format_filter	Character vector of image formats to include (e.g. c("png", "jpg")).
.search_recency_filter	Restrict search to recent results: "hour", "day", "week", "month", or "year".
.search_mode	Search index to use: "web" (default), "academic", or "sec".
.search_after_date_filter	Only include content published after this date (MM/DD/YYYY).
.search_before_date_filter	Only include content published before this date (MM/DD/YYYY).
.last_updated_after_filter	Only include content last updated after this date (MM/DD/YYYY).
.last_updated_before_filter	Only include content last updated before this date (MM/DD/YYYY).
.disable_search	Logical; if TRUE, disables web search entirely (default: FALSE).
.enable_search_classifier	Logical; if TRUE, lets the model decide whether to search (default: FALSE).
.reasoning_effort	Reasoning level: "low", "medium", or "high".

`.return_related_questions`
: Logical; if TRUE, returns related questions.

`.user_location`
: Named list for geographic search personalisation. Accepted fields: `country` (ISO 3166-1 alpha-2), `city`, `region`, `latitude`, `longitude`. Example: `list(country = "DE", city = "Berlin")`.

`.search_context_size`
: Amount of search context to include: "low", "medium" (default), or "high".

`.search_type`
: Search quality preference inside `web_search_options`: "fast", "pro", or "auto".

`.stream_mode`
: Response format: "full" (default) or "concise".

`.json_schema`
: A tidyllm schema created with `tidyllm_schema()` for structured JSON output (optional).

`.top_k`
: Top-k token sampling (integer, 0 disables).

`.web_search_options`
: Named list with raw web_search_options overrides. Values set here take precedence over the dedicated parameters above.

`.api_url`
: API endpoint (default: "https://api.perplexity.ai/").

`.timeout`
: Timeout in seconds (default: 60).

`.stream`
: If TRUE, streams response.

`.verbose`
: If TRUE, prints additional info.

`.max_tries`
: Max request retries (default: 3).

`.dry_run`
: If TRUE, returns constructed request instead of sending.

Value

An updated `LLMMessage` object with the assistant's reply and metadata, including citations and search_results.

Deprecated

Perplexity support is deprecated since tidyllm 0.7.0 and ends in 0.8.0. Use `openrouter(.model = "perplexity/sonar")` for the same models, or `websearch_tool()` to give any provider web search.

`perplexity_check_research`

Check the Status of a Perplexity Deep Research Job

Description

Check the Status of a Perplexity Deep Research Job

Usage

```
perplexity_check_research(
  .job,
  .api_key = Sys.getenv("PERPLEXITY_API_KEY"),
  .max_tries = 3
)
```

Arguments

.job	A tidyllm_research_job object returned by perplexity_deep_research(.background = TRUE).
.api_key	Character; Perplexity API key (default: from environment).
.max_tries	Integer; maximum retries (default: 3).

Value

An updated tidyllm_research_job with a $status field and $response if completed.

Deprecated

Perplexity support is deprecated since tidyllm 0.7.0 and ends in 0.8.0. Use openrouter(.model = "perplexity/sonar") for the same models, or websearch_tool() to give any provider web search.

---

perplexity_deep_research

Submit a Deep Research Request to Perplexity

Description

Submit a Deep Research Request to Perplexity

Usage

```
perplexity_deep_research(
  .llm,
  .background = FALSE,
  .reasoning_effort = "medium",
  .search_context_size = "medium",
  .search_domain_filter = NULL,
  .search_language_filter = NULL,
  .language_preference = NULL,
  .search_recency_filter = NULL,
  .search_mode = NULL,
  .search_after_date_filter = NULL,
  .search_before_date_filter = NULL,
```

```
  .last_updated_after_filter = NULL,
  .last_updated_before_filter = NULL,
  .user_location = NULL,
  .json_schema = NULL,
  .idempotency_key = NULL,
  .api_key = Sys.getenv("PERPLEXITY_API_KEY"),
  .timeout = 300,
  .max_tries = 3
)
```

Arguments

.llm	An LLMMessage object containing the research question.
.background	Logical; if TRUE, returns a tidyllm_research_job immediately without waiting (default: FALSE).
.reasoning_effort	Reasoning level: "low", "medium" (default), or "high".
.search_context_size	Amount of search context: "low", "medium" (default), or "high".
.search_domain_filter	Character vector of domains to allowlist/denylist (max 10; prefix with "-" to denylist).
.search_language_filter	ISO 639-1 language code to restrict search results (e.g. "en", "de").
.language_preference	ISO 639-1 code for preferred response language.
.search_recency_filter	Restrict search to recent results: "hour", "day", "week", "month", or "year".
.search_mode	Search index to use: "web" (default), "academic", or "sec".
.search_after_date_filter	Only include content published after this date (MM/DD/YYYY).
.search_before_date_filter	Only include content published before this date (MM/DD/YYYY).
.last_updated_after_filter	Only include content last updated after this date (MM/DD/YYYY).
.last_updated_before_filter	Only include content last updated before this date (MM/DD/YYYY).
.user_location	Named list for geographic search personalisation (fields: country, city, region, latitude, longitude).
.json_schema	A tidyllm schema created with tidyllm_schema() for structured JSON output (optional).
.idempotency_key	Optional string; unique key to prevent duplicate submissions.
.api_key	Character; Perplexity API key (default: from environment variable).
.timeout	Integer; request timeout in seconds for blocking polling (default: 300).
.max_tries	Integer; maximum retries (default: 3).

Value

If .background = FALSE, an updated LLMMessage with the research reply. If .background = TRUE, a tidyllm_research_job object.

Deprecated

Perplexity support is deprecated since tidyllm 0.7.0 and ends in 0.8.0. Use openrouter(.model = "perplexity/sonar") for the same models, or websearch_tool() to give any provider web search.

---

perplexity_fetch_research

Fetch Results from a Completed Perplexity Deep Research Job

Description

Fetch Results from a Completed Perplexity Deep Research Job

Usage

```
perplexity_fetch_research(
  .job,
  .api_key = Sys.getenv("PERPLEXITY_API_KEY"),
  .max_tries = 3
)
```

Arguments

.job	A tidyllm_research_job object. Must have status "completed" or will poll once.
.api_key	Character; Perplexity API key (default: from environment).
.max_tries	Integer; maximum retries (default: 3).

Value

An updated LLMMessage with the research reply appended.

Deprecated

Perplexity support is deprecated since tidyllm 0.7.0 and ends in 0.8.0. Use openrouter(.model = "perplexity/sonar") for the same models, or websearch_tool() to give any provider web search.

provider_capabilities *What each provider supports*

Description

Lists the verbs a provider implements and the arguments each verb accepts, or the media types it accepts in a message. Use it to find out which providers can do something before you write code that depends on it.

Usage

```
provider_capabilities(
  .provider = NULL,
  .verb = NULL,
  .argument = NULL,
  .what = c("arguments", "media"),
  .internal = FALSE
)
```

Arguments

.provider	A provider call such as claude(), a provider name such as "claude", or NULL (default) for every provider.
.verb	Character vector of verb names (such as "chat" or "send_batch") to keep. NULL keeps all verbs.
.argument	Character vector of argument names (such as ".thinking") to keep. NULL keeps all arguments.
.what	"arguments" (default) for verbs and arguments, or "media" for one row per provider and media type.
.internal	Logical; if TRUE, include the internal build verb that send_chat() and parallel_chat() use. Default FALSE.

Details

With .what = "arguments" the result has one row per provider, verb and argument. A verb that takes no arguments of its own gets one row with argument = NA. The default column holds the argument's default as text, so for .model it is the provider's default model. fn names the function that implements the verb for that provider; its help page documents every argument.

The table says that a provider's function accepts an argument. It does not say that every model of that provider accepts it: for example, .thinking can be accepted by openai() while a particular model rejects some effort levels.

Value

A tibble. For .what = "arguments": provider, verb, argument, default, fn. For .what = "media": provider, media (image, pdf, audio, video, files) and supported.

Examples

```
provider_capabilities(claude(), .verb = "chat")

provider_capabilities(.argument = ".thinking")

provider_capabilities(.what = "media")
```

`rate_limit_info` *Get the current rate limit information for all or a specific API*

Description

This function retrieves the rate limit details for the specified API, or for all APIs stored in the .tidyllm_rate_limit_env if no API is specified.

Usage

```
rate_limit_info(.api_name = NULL)
```

Arguments

`.api_name`	(Optional) The name of the API whose rate limit info you want to get If not provided, the rate limit info for all APIs in the environment will be returned

Value

A tibble containing the rate limit information.

`send_azure_openai_batch`
Send a Batch of Messages to Azure OpenAI Batch API

Description

This function creates and submits a batch of messages to the Azure OpenAI Batch API for asynchronous processing.

Usage

```
send_azure_openai_batch(
  .llms,
  .endpoint_url = Sys.getenv("AZURE_ENDPOINT_URL"),
  .deployment = "gpt-4o-mini",
  .api_version = "2024-10-01-preview",
  .max_completion_tokens = NULL,
  .reasoning_effort = NULL,
  .frequency_penalty = NULL,
  .logit_bias = NULL,
  .presence_penalty = NULL,
  .seed = NULL,
  .stop = NULL,
  .temperature = NULL,
  .top_p = NULL,
  .logprobs = NULL,
  .top_logprobs = NULL,
  .dry_run = FALSE,
  .overwrite = FALSE,
  .max_tries = 3,
  .timeout = 60,
  .verbose = FALSE,
  .json_schema = NULL,
  .id_prefix = "tidyllm_azure_openai_req_"
)
```

Arguments

.llms	A list of LLMMessage objects containing conversation histories.
.endpoint_url	Base URL for the API (default: Sys.getenv("AZURE_ENDPOINT_URL")).
.deployment	The identifier of the model that is deployed (default: "gpt-4o-mini").
.api_version	Which version of the API is deployed (default: "2024-10-01-preview")
.max_completion_tokens	Integer specifying the maximum tokens per response (default: NULL).
.reasoning_effort	How long should reasoning models reason (can either be "low","medium" or "high")
.frequency_penalty	Number between -2.0 and 2.0. Positive values penalize new tokens based on their existing frequency.
.logit_bias	A named list modifying the likelihood of specified tokens appearing in the completion.
.presence_penalty	Number between -2.0 and 2.0. Positive values penalize new tokens based on whether they appear in the text so far.
.seed	If specified, the system will make a best effort to sample deterministically.

.stop	Up to 4 sequences where the API will stop generating further tokens.
.temperature	What sampling temperature to use, between 0 and 2. Higher values make the output more random.
.top_p	An alternative to sampling with temperature, called nucleus sampling.
.logprobs	If TRUE, get the log probabilities of each output token (default: NULL).
.top_logprobs	If specified, get the top N log probabilities of each output token (0-5, default: NULL).
.dry_run	Logical; if TRUE, returns the prepared request object without executing it (default: FALSE).
.overwrite	Logical; if TRUE, allows overwriting an existing batch ID (default: FALSE).
.max_tries	Maximum number of retries to perform the request (default: 3).
.timeout	Integer specifying the request timeout in seconds (default: 60).
.verbose	Logical; if TRUE, additional info about the requests is printed (default: FALSE).
.json_schema	A JSON schema object provided by tidyllm schema or ellmer schemata (default: NULL).
.id_prefix	Character string to specify a prefix for generating custom IDs when names in .llms are missing (default: "tidyllm_azure_openai_req_").

Value

An updated and named list of .llms with identifiers that align with batch responses, including a batch_id attribute.

---

send_batch *Send a batch of messages to a batch API*

Description

The send_batch() function allows you to send a list of LLMMessage objects to an API. It routes the input to the appropriate provider-specific batch API function.

Usage

```
send_batch(
  .llms,
  .provider = getOption("tidyllm_sbatch_default"),
  .dry_run = NULL,
  .temperature = NULL,
  .timeout = NULL,
  .top_p = NULL,
  .max_tries = NULL,
  .model = NULL,
  .verbose = NULL,
  .json_schema = NULL,
```

```
  .seed = NULL,
  .stop = NULL,
  .frequency_penalty = NULL,
  .presence_penalty = NULL,
  .id_prefix = NULL
)
```

Arguments

.llms	A list of LLMMessage objects containing conversation histories.
.provider	A function or function call specifying the language model provider and any additional parameters. This should be a call to a provider function like openai(), claude(), etc. You can also set a default provider function via the tidyllm_sbatch_default option.
.dry_run	Logical; if TRUE, simulates the request without sending it to the provider. Useful for testing.
.temperature	Numeric; controls the randomness of the model's output (0 = deterministic).
.timeout	Numeric; the maximum time (in seconds) to wait for a response.
.top_p	Numeric; nucleus sampling parameter, which limits the sampling to the top cumulative probability p.
.max_tries	Integer; the maximum number of retries for failed requests.
.model	Character; the model identifier to use (e.g., "gpt-4").
.verbose	Logical; if TRUE, prints additional information about the request and response.
.json_schema	List; A JSON schema object as R list to enforce the output structure
.seed	Integer; sets a random seed for reproducibility.
.stop	Character vector; specifies sequences where the model should stop generating further tokens.
.frequency_penalty	Numeric; adjusts the likelihood of repeating tokens (positive values decrease repetition).
.presence_penalty	Numeric; adjusts the likelihood of introducing new tokens (positive values encourage novelty).
.id_prefix	Character string to specify a prefix for generating custom IDs when names in .llms are missing

Value

An updated and named list of .llms with identifiers that align with batch responses, including a batch_id attribute.

send_chat *Start a chat without waiting for it*

Description

Sends one request and returns immediately with a job handle. The session stays usable while the model works; use `check_job()` for its status, `get_partial()` for the text so far, and `fetch_job()` for the finished `LLMMessage`, which is exactly what `chat()` would have returned.

Usage

```
send_chat(
  .llm,
  .provider = getOption("tidyllm_chat_default"),
  .on_chunk = NULL,
  .stream = TRUE,
  .dry_run = NULL,
  .temperature = NULL,
  .timeout = NULL,
  .top_p = NULL,
  .max_tries = NULL,
  .model = NULL,
  .verbose = NULL,
  .json_schema = NULL,
  .tools = NULL,
  .max_tool_rounds = NULL,
  .seed = NULL,
  .stop = NULL,
  .frequency_penalty = NULL,
  .presence_penalty = NULL
)
```

Arguments

`.llm`	An `LLMMessage` object.
`.provider`	A provider function call, as in `chat()`.
`.on_chunk`	Optional function of one argument, called with each text delta as it arrives. This is the push form of a stream; `\(d) cat(d)` echoes to the console, and writing to a `reactiveVal` is all a Shiny app needs. Only meaningful with `.stream = TRUE`.
`.stream`	Logical; whether the provider streams the reply. Streaming is what makes `.on_chunk` and `get_partial()` show progress. A non-streaming job still runs without blocking; it simply has nothing to report until it finishes.
`.dry_run`	Logical; if `TRUE`, simulates the request without sending it to the provider. Useful for testing.

.temperature	Numeric; controls the randomness of the model's output (0 = deterministic).
.timeout	Numeric; the maximum time (in seconds) to wait for a response.
.top_p	Numeric; nucleus sampling parameter, which limits the sampling to the top cumulative probability p.
.max_tries	Integer; the maximum number of retries for failed requests.
.model	Character; the model identifier to use (e.g., "gpt-4").
.verbose	Logical; if TRUE, prints additional information about the request and response.
.json_schema	List; A JSON schema object as R list to enforce the output structure
.tools	Either a single TOOL object or a list of TOOL objects representing the available functions for tool calls.
.max_tool_rounds	Integer; the maximum number of tool use iterations for multi-turn tool calling (default varies by provider).
.seed	Integer; sets a random seed for reproducibility.
.stop	Character vector; specifies sequences where the model should stop generating further tokens.
.frequency_penalty	Numeric; adjusts the likelihood of repeating tokens (positive values decrease repetition).
.presence_penalty	Numeric; adjusts the likelihood of introducing new tokens (positive values encourage novelty).

Details

The request is driven from R's event loop rather than from a thread or a second process, so progress happens whenever the session yields, which the accessors do on your behalf. The consequence worth knowing is the other side of that: a blocking call of your own, a long `Sys.sleep()` or another `chat()`, pauses the job for its duration.

A streamed job is not retried after a transient 429 or 503 the way `chat()` is, because it is read from an open connection; a non-streamed one keeps the usual retries.

Requires the `later` package, and `promises` as well when `.stream = FALSE`.

Value

A `tidyllm_chat_job`.

Examples

```
## Not run:
job <- llm_message("Summarise the history of R in 500 words") |>
  send_chat(claude(), .stream = TRUE)

while (check_job(job) == "running") {
  cat("\r", nchar(get_partial(job)), "characters so far")
}
```

```
reply <- fetch_job(job)

## End(Not run)
```

---

send_claude_batch *Send a Batch of Messages to Claude API*

Description

This function creates and submits a batch of messages to the Claude API for asynchronous processing.

Usage

```
send_claude_batch(
  .llms,
  .model = "claude-sonnet-5-5",
  .max_tokens = 1024,
  .temperature = NULL,
  .top_k = NULL,
  .top_p = NULL,
  .stop_sequences = NULL,
  .json_schema = NULL,
  .thinking = FALSE,
  .thinking_budget = 1024,
  .effort = NULL,
  .cache = FALSE,
  .api_url = "https://api.anthropic.com/",
  .verbose = FALSE,
  .dry_run = FALSE,
  .overwrite = FALSE,
  .max_tries = 3,
  .timeout = 60,
  .id_prefix = "tidyllm_claude_req_"
)
```

Arguments

.llms	A list of LLMMessage objects containing conversation histories.
.model	Character string specifying the Claude model version (default: "claude-sonnet-5-5").
.max_tokens	Integer specifying the maximum tokens per response (default: 1024).
.temperature	Numeric between 0 and 1 controlling response randomness. Only supported on older models; Claude Sonnet 5 and Opus 4.7 or newer reject sampling parameters.

`.top_k` Integer for diversity by limiting the top K tokens. Only supported on older models.

`.top_p` Numeric between 0 and 1 for nucleus sampling. Only supported on older models.

`.stop_sequences` Character vector of sequences that halt response generation.

`.json_schema` A schema to enforce an output structure

`.thinking` Logical; if TRUE, enables Claude's thinking mode. On Claude Sonnet 4.6, Opus 4.6 or newer this maps to adaptive thinking; on older models it uses a fixed thinking budget (default: FALSE).

`.thinking_budget` Integer specifying the maximum tokens Claude can spend on thinking (default: 1024). Must be at least 1024. Only used on older models without adaptive thinking.

`.effort` Character; one of "low", "medium", "high", "xhigh", or "max". Controls thinking depth and overall token spend on models that support the effort parameter. Default NULL uses the API default.

`.cache` Logical or character; enables Anthropic prompt caching for the shared system prompt across the batch. TRUE caches with the default 5-minute time to live; "1h" requests a one-hour time to live. Useful when many requests share one large system prompt, provided that prompt clears the per-model minimum cacheable length (default: FALSE).

Defaults to "tidyllm_claude_req_".

`.api_url` Base URL for the Claude API (default: "https://api.anthropic.com/").

`.verbose` Logical; if TRUE, prints a message with the batch ID (default: FALSE).

`.dry_run` Logical; if TRUE, returns the prepared request object without executing it (default: FALSE).

`.overwrite` Logical; if TRUE, allows overwriting an existing batch ID associated with the request (default: FALSE).

`.max_tries` Maximum number of retries to perform the request.

`.timeout` Integer specifying the request timeout in seconds (default: 60).

`.id_prefix` Character string to specify a prefix for generating custom IDs when names in `.llms` are missing.

Value

An updated and named list of `.llms` with identifiers that align with batch responses, including a `batch_id` attribute.

send_gemini_batch *Submit a list of LLMMessage objects to Gemini's batch API*

Description

Returns a named list (same as input) with batch_id and json attributes.

Usage

```
send_gemini_batch(
  .llms,
  .model = "gemini-3.8-flash",
  .temperature = NULL,
  .max_output_tokens = NULL,
  .top_p = NULL,
  .top_k = NULL,
  .presence_penalty = NULL,
  .frequency_penalty = NULL,
  .stop_sequences = NULL,
  .safety_settings = NULL,
  .json_schema = NULL,
  .grounding_threshold = NULL,
  .thinking_level = NULL,
  .timeout = 120,
  .dry_run = FALSE,
  .max_tries = 3,
  .display = "tidyllm_batch",
  .id_prefix = "tidyllm_gemini_req_"
)
```

Arguments

.llms	List of LLMMessage objects (named or unnamed).
.model	The model identifier (default: "gemini-1.5-flash").
.temperature	Controls randomness (default: NULL, range: 0-2). Ignored by Gemini 3.6 Flash and later models; Google plans to reject it on upcoming models.
.max_output_tokens	Maximum tokens in the response (default: NULL).
.top_p	Nucleus sampling (default: NULL, range: 0-1). Ignored by Gemini 3.6 Flash and later models; Google plans to reject it on upcoming models.
.top_k	Diversity in token selection (default: NULL). Ignored by Gemini 3.6 Flash and later models; Google plans to reject it on upcoming models.
.presence_penalty	Penalizes new tokens (default: NULL, -2 to 2).

.frequency_penalty
: Penalizes frequent tokens (default: NULL, -2 to 2).

.stop_sequences
: Character vector or NULL of up to 5.

.safety_settings
: Optional list of safety settings (default: NULL).

.json_schema
: Optional schema to enforce output structure.

.grounding_threshold
: Optional grounding threshold (0-1) to enable Google Search.

.thinking_level
: How much the model reasons before answering: `"minimal"`, `"low"`, `"medium"` or `"high"` (default: NULL, the model's own default). Supported levels vary by model.

.timeout
: Timeout in seconds (default: 120).

.dry_run
: If TRUE, returns the constructed request (default: FALSE).

.max_tries
: Maximum retry attempts (default: 3).

.display
: Display name for this batch (default: "tidyllm_batch").

.id_prefix
: Prefix for message IDs (default: "tidyllm_gemini_req_").

Value

Named list of LLMMessage objects with attributes `batch_id` and `json`

`send_groq_batch` *Send a Batch of Messages to the Groq API*

Description

This function creates and submits a batch of messages to the Groq API for asynchronous processing.

Usage

```
send_groq_batch(
  .llms,
  .model = "openai/gpt-oss-120b",
  .max_tokens = 1024,
  .temperature = NULL,
  .top_p = NULL,
  .frequency_penalty = NULL,
  .presence_penalty = NULL,
  .stop = NULL,
  .seed = NULL,
  .api_url = "https://api.groq.com/",
  .json_schema = NULL,
  .completion_window = "24h",
```

```
  .verbose = FALSE,
  .dry_run = FALSE,
  .overwrite = FALSE,
  .max_tries = 3,
  .timeout = 60,
  .id_prefix = "tidyllm_groq_req_"
)
```

Arguments

`.llms`	A list of LLMMessage objects containing conversation histories.
`.model`	Character string specifying the model to use (default: "deepseek-r1-distill-llama-70b").
`.max_tokens`	Integer specifying the maximum tokens per response (default: 1024).
`.temperature`	Numeric between 0 and 2 controlling response randomness.
`.top_p`	Numeric between 0 and 1 for nucleus sampling.
`.frequency_penalty`	Number between -2.0 and 2.0 to penalize repetition.
`.presence_penalty`	Number between -2.0 and 2.0 to encourage new topics.
`.stop`	One or more sequences where the API will stop generating further tokens.
`.seed`	An integer for deterministic sampling.
`.api_url`	Base URL for the Groq API (default: "https://api.groq.com/").
`.json_schema`	A list or tidyllm schema created with `tidyllm_schema()` for structured JSON output (optional).
`.completion_window`	Character string for the batch completion window (default: "24h").
`.verbose`	Logical; if TRUE, prints a message with the batch ID (default: FALSE).
`.dry_run`	Logical; if TRUE, returns the prepared request objects without executing (default: FALSE).
`.overwrite`	Logical; if TRUE, allows overwriting an existing batch ID (default: FALSE).
`.max_tries`	Maximum number of retries to perform the request.
`.timeout`	Integer specifying the request timeout in seconds (default: 60).
`.id_prefix`	Character string to specify a prefix for generating custom IDs when names in `.llms` are missing.

Value

An updated and named list of `.llms` with identifiers that align with batch responses, including a `batch_id` attribute.

`send_mistral_batch` *Send a Batch of Requests to the Mistral API*

Description

This function creates and submits a batch of messages to the Mistral API for asynchronous processing.

Usage

```
send_mistral_batch(
  .llms,
  .model = "mistral-small-latest",
  .endpoint = "/v1/chat/completions",
  .metadata = NULL,
  .temperature = 0.7,
  .top_p = 1,
  .max_tokens = 1024,
  .min_tokens = NULL,
  .frequency_penalty = NULL,
  .logit_bias = NULL,
  .presence_penalty = NULL,
  .seed = NULL,
  .stop = NULL,
  .safe_prompt = FALSE,
  .json_schema = NULL,
  .reasoning_effort = NULL,
  .dry_run = FALSE,
  .overwrite = FALSE,
  .max_tries = 3,
  .timeout = 60,
  .id_prefix = "tidyllm_mistral_req_"
)
```

Arguments

`.llms`	A list of LLMMessage objects containing conversation histories.
`.model`	The Mistral model version (default: "mistral-small-latest").
`.endpoint`	The API endpoint (default: "/v1/chat/completions").
`.metadata`	Optional metadata for the batch.
`.temperature`	Sampling temperature to use, between `0.0` and `1.5` (default: `0.7`).
`.top_p`	Nucleus sampling parameter, between `0.0` and `1.0` (default: `1`).
`.max_tokens`	The maximum number of tokens to generate in the completion (default: `1024`).
`.min_tokens`	The minimum number of tokens to generate (optional).

.frequency_penalty
: Numeric value (or NULL) for frequency penalty.

.logit_bias
: A named list modifying the likelihood of specific tokens (or NULL).

.presence_penalty
: Numeric value (or NULL) for presence penalty.

.seed
: Random seed for deterministic outputs (optional).

.stop
: Sequence(s) at which to stop generation (optional).

.safe_prompt
: Logical; if TRUE, injects a safety prompt (default: FALSE).

.json_schema
: A JSON schema object for structured output (optional).

.reasoning_effort
: How much reasoning models think before answering; one of "none", "low", "medium", or "high" (default: NULL, meaning the API default). Supported values vary by model: mistral-large-4 accepts only "high" and "none".

.dry_run
: Logical; if TRUE, returns the prepared request without executing it (default: FALSE).

.overwrite
: Logical; if TRUE, allows overwriting existing custom IDs (default: FALSE).

.max_tries
: Maximum retry attempts for requests (default: 3).

.timeout
: Request timeout in seconds (default: 60).

.id_prefix
: Prefix for generating custom IDs (default: "tidyllm_mistral_req_").

Value

The prepared LLMMessage list with a batch_id attribute.

---

send_ollama_batch *Send a Batch of Messages to Ollama API*

Description

This function creates and submits a batch of messages to the Ollama API Contrary to other batch functions, this functions waits for the batch to finish and receives requests. The advantage compared to sending single messages via chat() is that Ollama handles large parallel requests quicker than many individual chat requests.

Usage

```
send_ollama_batch(
  .llms,
  .model = "qwen3.5:4b",
  .stream = FALSE,
  .seed = NULL,
  .json_schema = NULL,
  .temperature = NULL,
  .num_ctx = 32768,
```

```
  .num_predict = NULL,
  .top_k = NULL,
  .top_p = NULL,
  .min_p = NULL,
  .mirostat = NULL,
  .mirostat_eta = NULL,
  .mirostat_tau = NULL,
  .repeat_last_n = NULL,
  .repeat_penalty = NULL,
  .tfs_z = NULL,
  .stop = NULL,
  .ollama_server = "http://localhost:11434",
  .timeout = 120,
  .keep_alive = NULL,
  .dry_run = FALSE
)
```

Arguments

.llms	A list of LLMMessage objects containing conversation histories.
.model	Character string specifying the Ollama model to use (default: "qwen3-vl")
.stream	Logical; whether to stream the response (default: FALSE)
.seed	Integer; seed for reproducible generation (default: NULL)
.json_schema	A JSON schema object as R list to enforce the output structure (default: NULL)
.temperature	Float between 0-2; controls randomness in responses (default: NULL)
.num_ctx	Integer; sets the context window size (default: 32768)
.num_predict	Integer; maximum number of tokens to predict (default: NULL)
.top_k	Integer; controls diversity by limiting top tokens considered (default: NULL)
.top_p	Float between 0-1; nucleus sampling threshold (default: NULL)
.min_p	Float between 0-1; minimum probability threshold (default: NULL)
.mirostat	Integer (0,1,2); enables Mirostat sampling algorithm (default: NULL)
.mirostat_eta	Float; Mirostat learning rate (default: NULL)
.mirostat_tau	Float; Mirostat target entropy (default: NULL)
.repeat_last_n	Integer; tokens to look back for repetition (default: NULL)
.repeat_penalty	Float; penalty for repeated tokens (default: NULL)
.tfs_z	Float; tail free sampling parameter (default: NULL)
.stop	Character; custom stop sequence(s) (default: NULL)
.ollama_server	String; Ollama API endpoint (default: "http://localhost:11434")
.timeout	Integer; API request timeout in seconds (default: 120)
.keep_alive	Character; How long should the ollama model be kept in memory after request (default: NULL - 5 Minutes)
.dry_run	Logical; if TRUE, returns request object without execution (default: FALSE)

Details

The function provides extensive control over the generation process through various parameters:

- Temperature (0-2): Higher values increase creativity, lower values make responses more focused
- Top-k/Top-p: Control diversity of generated text
- Mirostat: Advanced sampling algorithm for maintaining consistent complexity
- Repeat penalties: Prevent repetitive text
- Context window: Control how much previous conversation is considered

Value

A list of updated `LLMMessage` objects, each with the assistant's response added if successful.

`send_openai_batch` *Send a Batch of Messages to OpenAI Batch API*

Description

This function creates and submits a batch of messages to the OpenAI Batch API for asynchronous processing.

Usage

```
send_openai_batch(
  .llms,
  .model = "gpt-6-luna",
  .max_completion_tokens = NULL,
  .reasoning_effort = NULL,
  .frequency_penalty = NULL,
  .logit_bias = NULL,
  .presence_penalty = NULL,
  .seed = NULL,
  .stop = NULL,
  .temperature = NULL,
  .top_p = NULL,
  .logprobs = NULL,
  .top_logprobs = NULL,
  .dry_run = FALSE,
  .overwrite = FALSE,
  .json_schema = NULL,
  .max_tries = 3,
  .timeout = 60,
  .verbose = FALSE,
  .id_prefix = "tidyllm_openai_req_"
)
```

Arguments

.llms	A list of LLMMessage objects containing conversation histories.
.model	Character string specifying the OpenAI model version (default: "gpt-6-luna").
.max_completion_tokens	Integer specifying the maximum tokens per response (default: NULL).
.reasoning_effort	How long should reasoning models reason (can either be "low","medium" or "high")
.frequency_penalty	Number between -2.0 and 2.0. Positive values penalize new tokens based on their existing frequency in the text so far.
.logit_bias	A named list modifying the likelihood of specified tokens appearing in the completion.
.presence_penalty	Number between -2.0 and 2.0. Positive values penalize new tokens based on whether they appear in the text so far.
.seed	If specified, the system will make a best effort to sample deterministically.
.stop	Up to 4 sequences where the API will stop generating further tokens.
.temperature	What sampling temperature to use, between 0 and 2. Higher values make the output more random.
.top_p	An alternative to sampling with temperature, called nucleus sampling.
.logprobs	If TRUE, get the log probabilities of each output token (default: NULL).
.top_logprobs	If specified, get the top N log probabilities of each output token (0-5, default: NULL).
.dry_run	Logical; if TRUE, returns the prepared request object without executing it (default: FALSE).
.overwrite	Logical; if TRUE, allows overwriting an existing batch ID associated with the request (default: FALSE).
.json_schema	A JSON schema object provided by tidyllm_schema or ellmer schemata (default: NULL).
.max_tries	Maximum number of retries to perform the request (default: 3).
.timeout	Integer specifying the request timeout in seconds (default: 60).
.verbose	Logical; if TRUE, additional info about the requests is printed (default: FALSE).
.id_prefix	Character string to specify a prefix for generating custom IDs when names in .llms are missing (default: "tidyllm_openai_req_").

Value

An updated and named list of .llms with identifiers that align with batch responses, including a batch_id attribute.

`tidyllm_example_app` *Run a tidyllm example Shiny app*

Description

The apps ship with the installed package, so they always match the version of tidyllm you have. Each one defaults to a local `ollama()` model and therefore runs with no API key and no spend; a dropdown switches to the cloud providers.

Usage

```
tidyllm_example_app(.app = NULL, ...)
```

Arguments

`.app`	Name of the app to run. With no argument the available apps are listed.
`...`	Passed to `shiny::runApp()`.

Details

Available apps:

`"model_explainer"`: fits a linear model to a small public dataset and streams two explanations of the coefficients side by side, one plain-English and one adversarial. Every number is computed in R; the model only narrates the coefficient table it is shown.

Value

Invisibly, the path to the app that was run.

Examples

```
## Not run:
tidyllm_example_app()
tidyllm_example_app("model_explainer")

## End(Not run)
```

`tidyllm_schema` *Create a JSON Schema for Structured Outputs*

Description

This function creates a JSON schema for structured outputs, supporting both character-based shorthand and S7 `tidyllm_field` objects. It also integrates with `ellmer` types like `ellmer::type_string()` if ellmer is in your namespace

Usage

```
tidyllm_schema(name = "tidyllm_schema", ...)
```

Arguments

`name`	A character string specifying the schema name (default: "tidyllm_schema").
`...`	Named arguments where each name represents a field, and each value is either a character string, a `tidyllm_field`, or an `ellmer` type. Supported character shorthand types: • "character" or "string" for character fields • "logical" for boolean fields • "numeric" for number fields • "factor(...)" for enumerations • Use `[]` to indicate vectors, e.g., "character[]"

Value

A list representing the JSON schema, suitable for use with `.json_schema` in LLM API calls.

Examples

```
## Not run:
# Example using different field types
address_schema <- tidyllm_schema(
  name = "AddressSchema",
  Street = field_chr("A common street name"),
  house_number = field_dbl(),
  City = field_chr("Name of a city"),
  State = field_fct("State abbreviation", .levels = c("CA", "TX", "Other")),
  Country = "string",
  PostalCode = "string"
)

llm_message("Imagine an address") |> chat(openai, .json_schema = address_schema)

# Example with vector field
tidyllm_schema(
```

```
  plz = field_dbl(.vector = TRUE)
)

## End(Not run)
```

`tidyllm_tool` *Create a Tool Definition for tidyllm*

Description

Creates a tool definition for use with Language Model API calls that support function calling. This function wraps an existing R function with schema information for LLM interaction.

Usage

```
tidyllm_tool(.f, .description = character(0), ...)
```

Arguments

`.f`	The function to wrap as a tool
`.description`	Character string describing what the tool does
`...`	Named arguments providing schema definitions for each function parameter using tidyllm_fields

Details

Each parameter schema in `...` should correspond to a parameter in the wrapped function. All required function parameters must have corresponding schema definitions.

Value

A `TOOL` class object that can be used with tidyllm `chat()` functions

Examples

```
get_weather <- function(location){}
weather_tool <- tidyllm_tool(
  get_weather,
  "Get the current weather in a given location",
  location = field_chr("The city and state, e.g., San Francisco, CA")
)
```

upload_file *Upload a File to a Provider's File Store*

Description

Upload a File to a Provider's File Store

Usage

```
upload_file(.provider, .path, ...)
```

Arguments

.provider	A provider function call (e.g. claude(), gemini(), openai()).
.path	Path to the local file to upload.
...	Additional provider-specific arguments.

Value

A tidyllm_file object.

vibe_cli *Chat through a locally installed Mistral Vibe CLI*

Description

vibe_cli() routes a chat to Mistral's vibe command line tool installed on your own machine. See vibe_cli_chat() for the arguments and for what happens to Vibe's own tools.

Usage

```
vibe_cli(..., .called_from = NULL)
```

Arguments

...	Arguments passed to the Vibe CLI chat function.
.called_from	Internal; the verb that dispatched here.

Value

The result of the requested action; for chat(), an updated LLMMessage.

Examples

```
## Not run:
llm_message("Explain R's S7 classes in three sentences.") |>
  chat(vibe_cli())

## End(Not run)
```

`vibe_cli_chat` *Chat with Mistral models through your own installed Mistral Vibe CLI*

Description

`vibe_cli()` runs Mistral's `vibe` command line tool installed on your machine and reads its JSON output back. Vibe uses the API key stored by `vibe --setup`, or your `MISTRAL_API_KEY`.

Usage

```
vibe_cli_chat(
  .llm,
  .model = NULL,
  .system_prompt_mode = "prepend",
  .cli_tools = FALSE,
  .max_turns = NULL,
  .max_budget_usd = NULL,
  .stateful = FALSE,
  .session_id = NULL,
  .binary = "vibe",
  .verbose = FALSE,
  .timeout = 300,
  .stream = FALSE,
  .dry_run = FALSE
)
```

Arguments

`.llm`	An `LLMMessage` object.
`.model`	Character; a Mistral model id such as "mistral-large-4". Default NULL uses the model Vibe is configured with.
`.system_prompt_mode`	One of "prepend" or "ignore". With "prepend" the message's system prompt is put before the conversation in the prompt text. With "ignore" it is dropped.
`.cli_tools`	Logical; FALSE, the default, refuses every tool call. TRUE hands over to Vibe's own configuration, which may let it run commands and edit files.
`.max_turns`	Integer; the most assistant turns Vibe may take. Default NULL leaves it to Vibe.

`.max_budget_usd`
: Numeric; stop the run once it has cost this many US dollars.

`.stateful`
: Logical; if TRUE, Vibe keeps the conversation on its side. The first call sends the conversation and records the session id in the metadata; later calls resume that session and send only the newest user message instead of replaying the history. Default FALSE sends the whole conversation every time.

`.session_id`
: Character; resume this Vibe session explicitly. Normally left NULL, because `.stateful = TRUE` picks the id up from the message history on its own.

`.binary`
: Character; the command to run, or a full path to it. Defaults to "vibe", which is looked for on the PATH and then in the places the installers use. If R does not find it, set `options(tidyllm_vibe_cli_path = "/path/to/vibe")` in your `.Rprofile`, or the `TIDYLLM_VIBE_CLI` environment variable.

`.verbose`
: Logical; if TRUE, prints rate limit information after the response.

`.timeout`
: Integer; seconds to wait for the whole run.

`.stream`
: Logical; accepted so that `send_chat()` works, but the reply always arrives in one piece, because Vibe does not stream it.

`.dry_run`
: Logical; if TRUE, returns the command that would be run instead of running it.

Details

Vibe is a coding agent, not a plain completion endpoint. By default tidyllm runs it in Vibe's ask-first mode, in which every tool call needs an approval that a non-interactive run cannot give, so Vibe can neither read nor change files. If the model still tries a tool, the refusal costs a turn; `.max_turns` caps how many it may spend. Set `.cli_tools = TRUE` to use Vibe's own configuration instead.

Vibe has no system prompt option, so a message's system prompt is put in front of the prompt text. Vibe does not report token counts, cost or the model in its output, so those metadata fields are `NA`, except the model when you set `.model`. Replies arrive in one piece; there is no streaming.

Value

A new `LLMMessage` object containing the original messages plus the reply from Vibe.

Examples

```
## Not run:
llm_message("What is R's S7 class system?") |>
  chat(vibe_cli())

llm_message("Explain vapply() in two sentences.") |>
  chat(vibe_cli(.model = "mistral-large-4"))

## End(Not run)
```

video_file *Create a Video Object*

Description

Stores a reference to a local video file for use in multimodal messages. The file is not encoded until the message is formatted for a provider.

Usage

```
video_file(.path)
```

Arguments

.path	The path to the video file on disk.

Value

A tidyllm_video object.

voyage *Voyage Provider Function*

Description

The voyage() function acts as a provider interface for interacting with the Voyage.ai API through tidyllm's verbs. It dynamically routes requests to voyage-specific functions. At the moment this is only voyage_embed()

Usage

```
voyage(..., .called_from = NULL)
```

Arguments

...	Parameters to be passed to the appropriate Voyage-specific function, such as model configuration, input text, or API-specific options.
.called_from	An internal argument specifying which action (e.g., embed) the function is invoked from. This argument is automatically managed by the tidyllm verbs and should not be modified by the user.

Value

The result of the requested action, depending on the specific function invoked

`voyage_embedding` *Generate Embeddings Using Voyage AI API*

Description

This function creates embedding vectors from text or multimodal inputs (text and images) using the Voyage AI API. It supports three types of input:

Usage

```
voyage_embedding(
  .input,
  .model = "voyage-4",
  .output_dimension = NULL,
  .timeout = 120,
  .dry_run = FALSE,
  .max_tries = 3,
  .verbose = FALSE
)
```

Arguments

`.input`	Input to embed. Can be: • A character vector of texts • An `LLMMessage` object (all textual components will be embedded) • A list containing a mix of character strings and `tidyllm_image` objects created with `img()`
`.model`	The embedding model identifier. For text-only: "voyage-3.5-lite" (default). For multimodal inputs: "voyage-multimodal-3.5" is used automatically.
`.output_dimension`	Optional integer to control output vector size (default: NULL, uses model default).
`.timeout`	Timeout for the API request in seconds (default: 120).
`.dry_run`	If TRUE, perform a dry run and return the request object without sending.
`.max_tries`	Maximum retry attempts for requests (default: 3).
`.verbose`	Should information about current rate limits be printed? (default: FALSE).

Details

1. Character vector: Embeds each text string separately
2. LLMMessage object: Extracts and embeds text content from messages
3. List of mixed content: Processes a combination of text strings and image objects created with `img()`

For multimodal inputs, the function automatically switches to Voyage's multimodal API and formats the response with appropriate labels (e.g., `"[IMG] image.png"`) for images.

Value

A tibble with two columns: `input` and `embeddings`.

- The `input` column contains the input texts or image labels
- The `embeddings` column is a list column where each row contains an embedding vector

Examples

```
## Not run:
# Text embeddings
voyage_embedding("How does photosynthesis work?")

# Multimodal embeddings
list("A banana", img("banana.jpg"), "Yellow fruit") |>
  voyage_embedding()

## End(Not run)
```

`voyage_rerank` *Rerank Documents Using Voyage AI API*

Description

Rerank Documents Using Voyage AI API

Usage

```
voyage_rerank(
  .query,
  .documents,
  .model = "rerank-3",
  .top_k = NULL,
  .api_key = Sys.getenv("VOYAGE_API_KEY"),
  .timeout = 60,
  .max_tries = 3
)
```

Arguments

`.query`	A single character string representing the search query.
`.documents`	A character vector of documents to rerank.
`.model`	The reranking model identifier (default: "rerank-3").
`.top_k`	Integer; return only the top-k results (default: NULL, returns all).
`.api_key`	Character; Voyage API key (default: from environment).
`.timeout`	Integer; request timeout in seconds (default: 60).
`.max_tries`	Integer; maximum retries (default: 3).

Value

A tibble with columns `index`, `document`, and `relevance_score`, sorted by score descending.

`websearch` *Search the Web*

Description

Runs one web search and returns the results as a tibble. Use it to collect search results as data, for example sources for a list of companies, or to see exactly what a model would receive from `websearch_tool()`, which takes the same arguments.

Usage

```
websearch(
  .query,
  .backend = c("tavily", "searxng"),
  .server = NULL,
  .max_results = 5,
  .include_content = FALSE,
  .max_chars = 4000,
  .timeout = 30,
  ...
)
```

Arguments

`.query`	The search query, a single string of at least two characters.
`.backend`	The search service to use: `"tavily"` (the default) or `"searxng"`. See the section on search services below.
`.server`	The address of your SearXNG server, such as `"http://localhost:8888"`. If `NULL`, the `SEARXNG_SERVER` environment variable is used. Only for `.backend = "searxng"`.
`.max_results`	The most results a search returns, between 1 and 20. SearXNG returns one page of results, which may hold fewer.
`.include_content`	If `TRUE`, each result also carries the text of the page, cut to `.max_chars` characters. Only Tavily can do this.
`.max_chars`	The maximum number of characters of page text per result when `.include_content = TRUE`. Use `Inf` to keep the whole page.
`.timeout`	Seconds to wait for each request to the search service. Busy or failing services are asked up to three times, for at most a minute in total.
`...`	Further search options passed to the search service by name. The options each service accepts are listed in the section on search services below; a misspelled option is an error.

Details

A failed search, for example with a wrong Tavily key, used-up credits or an unreachable SearXNG server, stops with an error. To search for many queries, map over them and bind the results; the `query` column keeps them apart.

Value

A tibble with one row per result and the columns `query`, `title`, `url`, `published` (a date, `NA` where the service does not know it), `snippet` and `text` (`NA` unless `.include_content = TRUE`). If the service writes a short summary (Tavily with `include_answer = TRUE`, or a SearXNG instant answer), it is attached as the attribute `"answer"`. If some SearXNG engines did not answer, their names and reasons are attached as the attribute `"unresponsive_engines"`.

Search services

Tavily is a paid search API with a free plan of 1,000 credits per month, no credit card needed; one basic search costs one credit. It needs a `TAVILY_API_KEY` environment variable. Options: `search_depth` (`"basic"`, `"advanced"`, `"fast"` or `"ultra-fast"`; `"advanced"` costs two credits), `topic` (`"general"`, `"news"` or `"finance"`), `time_range` (`"day"`, `"week"`, `"month"` or `"year"`, or the short forms `"d"`, `"w"`, `"m"` and `"y"`), `start_date` and `end_date` (`"YYYY-MM-DD"`), `include_domains`, `exclude_domains`, `country`, `include_answer` and others from Tavily's search API.

SearXNG is a free search engine you run yourself, which collects results from Google, Brave and other engines. Use your own server: public SearXNG servers usually refuse programs. Its `settings.yml` must list `json` under `search: formats:`, and for a server only you use, `server: limiter: false` stops it from blocking repeated searches. Set the address with `.server` or the `SEARXNG_SERVER` environment variable. SearXNG returns no page text. Options: `categories` (such as `"general"` or `"news"`), `engines` (such as `c("google", "brave")`), `language` (such as `"de"`), `time_range` (`"day"`, `"week"`, `"month"` or `"year"`), `safesearch` (`0`, `1` or `2`) and `pageno` (which page of results, for more than one page). A search fails if it finds nothing and at least one engine did not answer, for example because it asked for a CAPTCHA.

Examples

```
## Not run:
websearch("tidyllm R package")

websearch("EU AI Act", .max_results = 10, topic = "news", time_range = "month")

websearch("EU AI Act", .backend = "searxng", .server = "http://localhost:8888",
          categories = "news", time_range = "week")

c("ZEW Mannheim", "ifo Institut") |>
  purrr::map(websearch, .max_results = 3) |>
  purrr::list_rbind()

## End(Not run)
```

websearch_tool *Give a Model Web Search*

Description

Creates a tool that lets any model search the web while it answers. Pass it to `chat()` with `.tools`, and the model decides when to search and what to search for. It works the same way with every provider that supports tools, including local models through `ollama()` or `llamacpp()` that have no web access of their own.

The model only chooses the search query. Everything else, such as how many results come back and whether full page text is included, is fixed when you create the tool, so a model cannot run up your search bill by asking for more. The arguments are the same as for `websearch()`, which runs a single search directly and returns the results as a tibble.

Usage

```
websearch_tool(
  .backend = c("tavily", "searxng"),
  .server = NULL,
  .max_results = 5,
  .include_content = FALSE,
  .max_chars = 4000,
  .timeout = 30,
  ...
)
```

Arguments

`.backend`	The search service to use: `"tavily"` (the default) or `"searxng"`. See the section on search services below.
`.server`	The address of your SearXNG server, such as `"http://localhost:8888"`. If `NULL`, the `SEARXNG_SERVER` environment variable is used. Only for `.backend = "searxng"`.
`.max_results`	How many results each search returns, between 1 and 20.
`.include_content`	If `TRUE`, each result also carries the text of the page, cut to `.max_chars` characters. This helps with questions a short excerpt cannot answer, but it makes every tool result much longer. Only Tavily can do this.
`.max_chars`	The maximum number of characters of page text per result when `.include_content = TRUE`. Use `Inf` to keep the whole page.
`.timeout`	Seconds to wait for each request to the search service. Busy or failing services are asked up to three times, for at most a minute in total.
`...`	Further search options passed to the search service by name. The options each service accepts are listed in the section on search services below; a misspelled option is an error.

Details

Each search returns one block of text to the model: the query, the date of the search, and a numbered list of results with title, URL, publication date where the service knows it, and an excerpt. The tool asks the model to cite the URLs it relies on.

If a search fails, for example because the key is wrong or the monthly credits are used up, the model receives the error message as the search result instead of the conversation stopping. It will usually tell you what went wrong.

The tool is named `tidyllm_web_search`, which is the name you will see in the tool calls of a conversation. The Tavily key or SearXNG address is read once, when the tool is created: set it before calling `websearch_tool()`, and create the tool again after changing it. The key is stored inside the tool, so do not save the tool object to a file you share.

Value

A tool object to pass to the `.tools` argument of `chat()`.

Search services

Tavily is a paid search API with a free plan of 1,000 credits per month, no credit card needed; one basic search costs one credit. It needs a `TAVILY_API_KEY` environment variable. Options: `search_depth` (`"basic"`, `"advanced"`, `"fast"` or `"ultra-fast"`; `"advanced"` costs two credits), `topic` (`"general"`, `"news"` or `"finance"`), `time_range` (`"day"`, `"week"`, `"month"` or `"year"`, or the short forms `"d"`, `"w"`, `"m"` and `"y"`), `start_date` and `end_date` (`"YYYY-MM-DD"`), `include_domains`, `exclude_domains`, `country`, `include_answer` and others from Tavily's search API.

SearXNG is a free search engine you run yourself, which collects results from Google, Brave and other engines. Use your own server: public SearXNG servers usually refuse programs. Its `settings.yml` must list `json` under `search: formats:`, and for a server only you use, `server: limiter: false` stops it from blocking repeated searches. Set the address with `.server` or the `SEARXNG_SERVER` environment variable. SearXNG returns no page text. Options: `categories` (such as `"general"` or `"news"`), `engines` (such as `c("google", "brave")`), `language` (such as `"de"`), `time_range` (`"day"`, `"week"`, `"month"` or `"year"`), `safesearch` (`0`, `1` or `2`) and `pageno` (which page of results, for more than one page). A search fails if it finds nothing and at least one engine did not answer, for example because it asked for a CAPTCHA.

Examples

```
## Not run:
llm_message("What changed in the latest R release?") |>
  chat(ollama(), .tools = websearch_tool())

news_search <- websearch_tool(.max_results = 8, topic = "news", time_range = "week")
llm_message("Summarise this week's news on EU AI regulation.") |>
  chat(claude(), .tools = news_search)

## End(Not run)
```

Index

∗ **Message Creation Utilities**
    df_llm_message, 36
    llm_message, 78

audio_file, 5
azure_openai, 6
azure_openai_chat, 6
azure_openai_embedding, 8

cancel_job, 9
cancel_openai_batch, 10
chat, 10
chat(), *100*, *115*
chat_completions
        *(*chat_completions_chat*)*, 12
chat_completions_chat, 12
chat_ellmer, 13
check_azure_openai_batch, 14
check_batch, 15
check_claude_batch, 15
check_gemini_batch, 16
check_groq_batch, 17
check_job, 18
check_job(), *44*, *115*
check_mistral_batch, 18
check_openai_batch, 19
claude, 20
claude_chat, 20
claude_cli, 23
claude_cli_chat, 23
claude_cli_chat(), *23*
claude_delete_file, 26
claude_file_metadata, 26
claude_list_files, 27
claude_list_models, 28
claude_upload_file, 28
claude_websearch, 29
codex_cli, 30
codex_cli_chat, 31
codex_cli_chat(), *30*
deep_research, 35
deepseek, 33
deepseek_chat, 33
delete_file, 36
df_llm_message, 36, *79*
df_llm_message(), *79*

ellmer, 37
ellmer_tool, 37
embed, 39

fetch_azure_openai_batch, 40
fetch_batch, 41
fetch_claude_batch, 42
fetch_gemini_batch, 43
fetch_groq_batch, 43
fetch_job, 44
fetch_job(), *115*
fetch_mistral_batch, 45
fetch_openai_batch, 45
field_chr, 46
field_dbl *(*field_chr*)*, 46
field_fct *(*field_chr*)*, 46
field_lgl *(*field_chr*)*, 46
field_object, 47
file_info, 48

gemini, 48
gemini_chat, 49
gemini_delete_file, 51
gemini_embedding, 51
gemini_file_metadata, 52
gemini_list_files, 52
gemini_list_models, 53
gemini_upload_file, 53
get_logprobs, 54
get_metadata, 55
get_metadata(), *54*
get_partial, 56
get_partial(), *58*, *115*

get_reply, 56
get_reply(), *57*
get_reply_data, 57
get_reply_data(), *56, 57*
get_stream, 58
get_user_message, 59
groq, 59
groq_chat, 60
groq_list_models, 62
groq_transcribe, 63

img, 64

last_metadata *(*get_metadata*)*, 55
last_metadata(), *55*
last_reply *(*get_reply*)*, 56
last_reply(), *56, 57*
last_reply_data *(*get_reply_data*)*, 57
last_reply_data(), *57*
last_user_message *(*get_user_message*)*, 59
last_user_message(), *59*
list_azure_openai_batches, 64
list_batches, 65
list_claude_batches, 65
list_files, 66
list_gemini_batches, 67
list_groq_batches, 67
list_hf_gguf_files, 68
list_mistral_batches, 68
list_models, 69
list_openai_batches, 70
llamacpp, 70
llamacpp_chat, 71
llamacpp_delete_model, 73
llamacpp_download_model, 73
llamacpp_embedding, 74
llamacpp_health, 75
llamacpp_list_local_models, 75
llamacpp_list_models, 76
llamacpp_rerank, 76
llm_message, *37*, 78
llm_message(), *37*
LLMMessage, 77

mistral, 79
mistral_chat, 80
mistral_embedding, 82
mistral_list_models, 82

ollama, 83
ollama_chat, 84
ollama_delete_model, 86
ollama_download_model, 87
ollama_embedding, 87
ollama_list_models, 88
openai, 88
openai_chat, 89
openai_check_research, 90
openai_code_interpreter, 91
openai_deep_research, 91
openai_embedding, 92
openai_fetch_research, 93
openai_list_models, 93
openai_websearch, 94
openrouter, 94
openrouter_chat, 95
openrouter_credits, 97
openrouter_embedding, 97
openrouter_generation, 98
openrouter_list_models, 99

parallel_chat, 99
parallel_chat(), *103*
pdf_file, 101
pdf_page_batch, 102
perplexity, 103
perplexity_chat, 104
perplexity_check_research, 106
perplexity_deep_research, 107
perplexity_fetch_research, 109
provider_capabilities, 110

rate_limit_info, 111

send_azure_openai_batch, 111
send_batch, 113
send_chat, 115
send_chat(), *9*, *56*, *58*, *101*
send_claude_batch, 117
send_gemini_batch, 119
send_groq_batch, 120
send_mistral_batch, 122
send_ollama_batch, 123
send_openai_batch, 125
shiny::runApp(), *127*

tidyllm_example_app, 127
tidyllm_schema, 128
tidyllm_tool, 129

upload_file, 130

vibe_cli, 130
vibe_cli_chat, 131
vibe_cli_chat(), *130*
video_file, 133
voyage, 133
voyage_embedding, 134
voyage_rerank, 135

websearch, 136
websearch(), *138*
websearch_tool, 138
websearch_tool(), *103*, *106*, *107*, *109*, *136*