Package 'tidyllm'

March 8, 2025

Title Tidy Integration of Large Language Models

Version 0.3.2

Description A tidy interface for integrating large language model (LLM) APIs such as 'Claude', 'Openai', 'Groq','Mistral' and local models via 'Ollama' into R workflows. The package supports text and media-based interactions, interactive message history, batch request APIs, and a tidy, pipeline-oriented interface for streamlined integration into data workflows. Web services are available at <https://www.anthropic.com>, <https://openai.com>, <https://groq.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 7.3.2

VignetteBuilder knitr

Suggests knitr, rmarkdown, testthat (>= 3.0.0), tidyverse, httptest2, httpuv, ellmer

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

Collate 'tidyllm-package.R' 'utilites.R' 'embedding_helpers.R' 'LLMMessage.R' 'APIProvider.R' 'llm_message.R' 'llm_verbs.R' 'media.R' 'message_retrieval.R' 'perform_api_requests.R' 'rate_limits.R' 'tidyllm_schema.R' 'tools.R' 'pdfbatch.R' 'api_openai.R' 'api_claude.R' 'api_gemini.R' 'api_ollama.R' 'api_azure_openai.R' 'api_groq.R' 'api_mistral.R' 'api_perplexity.R' 'api_deepseek.R' 'api_voyage.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 2025-03-08 00:40:05 UTC

Contents

`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 OpenAI Chat Completions endpoint on Azure*

Description

This function sends a message history to the Azure OpenAI Chat Completions API and returns the assistant's reply. This function is work in progress and not fully tested

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,
  .frequency_penalty = NULL,
  .logit_bias = NULL,
  .logprobs = FALSE,
  .top_logprobs = NULL,
  .presence_penalty = NULL,
  .seed = NULL,
  .stop = NULL,
  .stream = FALSE,
  .temperature = NULL,
  .top_p = NULL,
  .timeout = 60,
  .verbose = FALSE,
  .json_schema = NULL,
  .dry_run = FALSE,
  .max_tries = 3,
  .tools = NULL,
  .tool_choice = NULL
)
```

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: "2024-10-01-preview")
`.max_completion_tokens`	An upper bound for the number of tokens that can be generated for a completion, including visible output tokens and reasoning tokens.
`.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.
`.logprobs`	Whether to return log probabilities of the output tokens (default: FALSE).
`.top_logprobs`	An integer between 0 and 20 specifying the number of most likely tokens to return at each token position.
`.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 as R list to enforce the output structure (If defined has precedence over JSON mode).
`.dry_run`	If TRUE, perform a dry run and return the request object (default: FALSE).
`.max_tries`	Maximum retries to perform request
`.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".

Value

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

Examples

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

# With custom parameters
result2 <- azure_openai_chat(msg,
                 .deployment = "gpt-4o-mini",
                 .temperature = 0.7,
                 .max_tokens = 1000)

## End(Not run)
```

`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 `LLMMesssage`object
`.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_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,
  .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.
`.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)
```

chatgpt *Alias for the OpenAI Provider Function*

Description

The `chatgpt` function is an alias for the `openai()` provider function. It provides a convenient way to interact with the OpenAI API for tasks such as sending chat messages, generating embeddings, and handling batch operations using `tidyllm` verbs like `chat()`, `embed()`, and `send_batch()`.

Usage

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

Arguments

`...`	Parameters to be passed to the appropriate OpenAI-specific function, such as model configuration, input text, or other API-specific options.
`.called_from`	An internal argument that specifies the context (e.g., `chat`, `embed`, `send_batch`) in which the function is being invoked. This 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()`).

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_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-3-5-sonnet-20241022",
  .max_tokens = 1024,
  .temperature = NULL,
  .top_k = NULL,
  .top_p = NULL,
  .metadata = NULL,
  .stop_sequences = NULL,
  .tools = NULL,
  .api_url = "https://api.anthropic.com/",
  .verbose = FALSE,
  .max_tries = 3,
  .timeout = 60,
  .stream = FALSE,
  .dry_run = FALSE
)
```

Arguments

`.llm`	An LLMMessage object containing the conversation history and system prompt.
`.model`	Character string specifying the Claude model version (default: "claude-3-5-sonnet-20241022").
`.max_tokens`	Integer specifying the maximum number of tokens in the response (default: 1024).

`.temperature`	Numeric between 0 and 1 controlling response randomness.
`.top_k`	Integer controlling diversity by limiting the top K tokens.
`.top_p`	Numeric between 0 and 1 for nucleus sampling.
`.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.
`.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).

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 custom parameters
result2 <- claude_chat(msg,
                 .temperature = 0.7,
                 .max_tokens = 1000)

## End(Not run)
```

`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.

`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-chat",
  .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
)
```

Arguments

`.llm`	An `LLMMessage` object containing the conversation history.
`.model`	The identifier of the model to use (default: "deepseek-chat").
`.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).

Value

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

`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()`

`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_ids` 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_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()` 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 )
```

`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-2.0-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,
  .timeout = 120,
  .dry_run = FALSE,
  .max_tries = 3,
  .verbose = FALSE,
  .stream = FALSE
)
```

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).
`.max_output_tokens`	Maximum tokens in the response (default: NULL).
`.top_p`	Controls nucleus sampling (default: NULL, range: 0.0-1.0).
`.top_k`	Controls diversity in token selection (default: NULL, range: 0 or more).
`.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.
`.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).

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 = "text-embedding-004",
  .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: "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).

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_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. Currently only works with `openai_chat()`

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).
- `api_specific`: A list column with API-specific metadata.

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_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_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 = "deepseek-r1-distill-llama-70b",
  .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 = FALSE,
  .timeout = 60,
  .verbose = FALSE,
  .stream = FALSE,
  .dry_run = FALSE,
  .max_tries = 3
)
```

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`	Whether the response should be structured as JSON (default: FALSE).
`.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

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.

Usage

```
img(.path)
```

Arguments

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

Value

An `tidyllm_image`, containing:

- `imagepath`: The original file path
- `imagename`: The basename of the image
- `imagebase64`: a "data:image/...;base64,..." string

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_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).

`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",
  .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.
.imagefile	Path to an image file to be attached (optional).
.pdf	Path to a PDF file to be attached (optional). Can be a character vector of length one (file path), or a list with filename, start_page, and end_page.

`.textfile`	Path to a text file to be read and attached (optional).
`.capture_plot`	Boolean to indicate whether a plot should be captured and attached as an image (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 = "mistral-large-latest",
  .stream = FALSE,
  .seed = NULL,
  .json = FALSE,
  .temperature = 0.7,
  .top_p = 1,
  .stop = NULL,
  .safe_prompt = FALSE,
  .timeout = 120,
  .max_tries = 3,
  .max_tokens = 1024,
  .min_tokens = NULL,
  .dry_run = FALSE,
  .verbose = FALSE,
  .tools = NULL,
  .tool_choice = NULL
)
```

Arguments

`.llm`	An `LLMMessage` object.
`.model`	The model identifier to use (default: `"mistral-large-latest"`).
`.stream`	Whether to stream back partial progress to the console. (default: `FALSE`).
`.seed`	The seed to use for random sampling. If set, different calls will generate deterministic results (optional).
`.json`	Whether the output should be in JSON mode(default: `FALSE`).
`.temperature`	Sampling temperature to use, between `0.0` and `1.5`. Higher values make the output more random, while lower values make it more focused and deterministic (default: `0.7`).
`.top_p`	Nucleus sampling parameter, between `0.0` and `1.0`. The model considers tokens with top_p probability mass (default: `1`).
`.stop`	Stop generation if this token is detected, or if one of these tokens is detected when providing a list (optional).
`.safe_prompt`	Whether to inject a safety prompt before all conversations (default: `FALSE`).

`.timeout`	When should our connection time out in seconds (default: `120`).
`.max_tries`	Maximum retries to peform request
`.max_tokens`	The maximum number of tokens to generate in the completion. Must be `>= 0` (default: `1024`).
`.min_tokens`	The minimum number of tokens to generate in the completion. Must be `>= 0` (optional).
`.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".

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 = "gemma2",
  .stream = FALSE,
  .seed = NULL,
  .json_schema = NULL,
  .temperature = NULL,
  .num_ctx = 2048,
  .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,
  .tfs_z = NULL,
  .stop = NULL,
  .ollama_server = "http://localhost:11434",
  .timeout = 120,
  .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: "gemma2")
`.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: 2048)
`.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.
`.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 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 = "all-minilm",
  .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` *OpenAI Provider Function*

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()`. It dynamically routes requests to OpenAI-specific functions like `openai_chat()` and `openai_embedding()` based on the context of the call.

Usage

```
openai(..., .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`, `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()`).

`openai_chat` *Send LLM Messages to the OpenAI Chat Completions API*

Description

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

Usage

```
openai_chat(
  .llm,
  .model = "gpt-4o",
  .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,
  .api_url = "https://api.openai.com/",
  .timeout = 60,
  .verbose = FALSE,
  .json_schema = NULL,
  .max_tries = 3,
  .dry_run = FALSE,
  .compatible = FALSE,
  .api_path = "/v1/chat/completions",
  .logprobs = NULL,
  .top_logprobs = NULL,
  .tools = NULL,
  .tool_choice = NULL
)
```

Arguments

`.llm`	An `LLMMessage` object containing the conversation history.
`.model`	The identifier of the model to use (default: "gpt-4o").

`.max_completion_tokens`	An upper bound for the number of tokens that can be generated for a completion, including visible output tokens and reasoning tokens.
`.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.
`.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.
`.api_url`	Base URL for the API (default: "https://api.openai.com/").
`.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).
`.compatible`	If TRUE, skip API and rate-limit checks for OpenAI compatible APIs (default: FALSE).
`.api_path`	The path relative to the base `.api_url` for the API (default: "/v1/chat/completions").
`.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".

Value

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

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_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.

`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 list of LLMMessage objects, each containing the text and image for a page.

`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()`).

perplexity_chat *Send LLM Messages to the Perplexity Chat API*

Description

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

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,
  .return_images = FALSE,
  .search_recency_filter = NULL,
  .api_url = "https://api.perplexity.ai/",
  .json = FALSE,
  .timeout = 60,
  .verbose = FALSE,
  .stream = FALSE,
  .dry_run = FALSE,
  .max_tries = 3
)
```

Arguments

`.llm`	An `LLMMessage` object containing the conversation history.
`.model`	The identifier of the model to use (default: "sonar").
`.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 (exclusive) and 2 (exclusive) are allowed, where higher values increase randomness (optional).
`.top_p`	Nucleus sampling parameter that controls the proportion of probability mass considered. Values between 0 (exclusive) and 1 (exclusive) are allowed (optional).
`.frequency_penalty`	Number greater than 0. Values > 1.0 penalize repeated tokens, reducing the 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).
`.search_domain_filter`	A vector of domains to limit or exclude from search results. For exclusion, prefix domains with a "-" (optional, currently in closed beta).
`.return_images`	Logical; if TRUE, enables returning images from the model's response (default: FALSE, currently in closed beta).
`.search_recency_filter`	Limits search results to a specific time interval (e.g., "month", "week", "day", or "hour"). Only applies to online models (optional).
`.api_url`	Base URL for the Perplexity API (default: "https://api.perplexity.ai/").
`.json`	Whether the response should be structured as JSON (default: FALSE).
`.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 perform the request (default: 3).

Value

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

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

Arguments

`.llms`	An `LLMMessage` object containing the conversation history.
`.deployment`	The identifier of the model that is deployed (default: "gpt-4o-mini").
`.endpoint_url`	Base URL for the API (default: Sys.getenv("AZURE_ENDPOINT_URL")).
`.api_version`	Which version of the API is deployed (default: "2024-10-01-preview")
`.max_completion_tokens`	An upper bound for the number of tokens that can be generated for a completion, including visible output tokens and reasoning tokens.

`.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.
`.logprobs`	Whether to return log probabilities of the output tokens (default: FALSE).
`.top_logprobs`	An integer between 0 and 20 specifying the number of most likely tokens to return at each token position.
`.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.
`.dry_run`	If TRUE, perform a dry run and return the request object (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`	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 as R list to enforce the output structure (default: NULL).
`.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.

`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_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-3-5-sonnet-20241022",
  .max_tokens = 1024,
  .temperature = NULL,
  .top_k = NULL,
  .top_p = NULL,
  .stop_sequences = NULL,
  .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-3-5-sonnet-20241022").
`.max_tokens`	Integer specifying the maximum tokens per response (default: 1024).
`.temperature`	Numeric between 0 and 1 controlling response randomness.
`.top_k`	Integer for diversity by limiting the top K tokens.
`.top_p`	Numeric between 0 and 1 for nucleus sampling.
`.stop_sequences`	Character vector of sequences that halt response generation.

`.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. Defaults to "tidyllm_claude_req_".

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

Send a Batch of Requests to the Mistral API

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,
  .seed = NULL,
  .stop = 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`. Higher values make the output more random (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).
`.seed`	Random seed for deterministic outputs (optional).
`.stop`	Stop generation at specific tokens or strings (optional).
`.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_llms` list with the `batch_id` attribute attached.

`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 = "gemma2",
  .stream = FALSE,
  .seed = NULL,
  .json_schema = NULL,
  .temperature = NULL,
  .num_ctx = 2048,
  .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: "gemma2")
`.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: 2048)
`.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-4o",
  .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-4o").
`.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_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")
)
```

`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-3",
  .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" (default). For multimodal inputs: "voyage-multimodal-3" is used automatically.
`.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)
```

Index