Package 'gptstudio'

July 22, 2025

Type Package

Title Use Large Language Models Directly in your Development Environment

Version 0.4.0

Maintainer James Wade <github@jameshwade.com>

Description Large language models are readily accessible via API. This package lowers the barrier to use the API inside of your development environment. For more on the API, see <https://platform.openai.com/docs/introduction>.

License MIT + file LICENSE

URL https://github.com/MichelNivard/gptstudio, https://michelnivard.github.io/gptstudio/

BugReports https://github.com/MichelNivard/gptstudio/issues

Depends R (>= 4.0)

Imports assertthat, bslib (>= 0.6.0), cli, colorspace, curl, fontawesome, glue, grDevices, htmltools, htmlwidgets, httr2, ids, jsonlite, magrittr, purrr, R6, rlang, rstudioapi (>= 0.12), rvest, shiny, shiny.i18n, SSEparser, stringr (>= 1.5.0), utils, waiter, yaml

Suggests AzureRMR, knitr, mockr, rmarkdown, shinytest2, spelling, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Language en-US

RoxygenNote 7.3.1

VignetteBuilder knitr

NeedsCompilation no

Author Michel Nivard [aut, cph],
James Wade [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0002-9740-1905>),
Samuel Calderon [aut] (ORCID: <https://orcid.org/0000-0001-6847-1210>)

Repository CRAN

Date/Publication 2024-05-21 11:21:21 UTC

Contents

`chat` *Chat Interface for gptstudio*

Description

This function provides a high-level interface for communicating with various services and models supported by gptstudio. It orchestrates the creation, configuration, and execution of a request based on user inputs and options set for gptstudio. The function supports a range of tasks from text generation to code synthesis and can be customized according to skill level and coding style preferences.

Usage

```
chat(
  prompt,
  service = getOption("gptstudio.service"),
  history = list(list(role = "system", content = "You are an R chat assistant")),
  stream = FALSE,
  model = getOption("gptstudio.model"),
  skill = getOption("gptstudio.skill"),
  style = getOption("gptstudio.code_style", "no preference"),
  task = getOption("gptstudio.task", "coding"),
  custom_prompt = NULL,
  process_response = FALSE,
```

```
  ...
)
```

Arguments

prompt	A string containing the initial prompt or question to be sent to the model. This is a required parameter.
service	The AI service to be used for the request. If not explicitly provided, this defaults to the value set in `getOption("gptstudio.service")`. If the option is not set, make sure to provide this parameter to avoid errors.
history	An optional parameter that can be used to include previous interactions or context for the current session. Defaults to a system message indicating "You are an R chat assistant".
stream	A logical value indicating whether the interaction should be treated as a stream for continuous interactions. If not explicitly provided, this defaults to the value set in `getOption("gptstudio.stream")`.
model	The specific model to use for the request. If not explicitly provided, this defaults to the value set in `getOption("gptstudio.model")`.
skill	A character string indicating the skill or capability level of the user. This parameter allows for customizing the behavior of the model to the user. If not explicitly provided, this defaults to the value set in `getOption("gptstudio.skill")`.
style	The coding style preferred by the user for code generation tasks. This parameter is particularly useful when the task involves generating code snippets or scripts. If not explicitly provided, this defaults to the value set in `getOption("gptstudio.code_style")`.
task	The specific type of task to be performed, ranging from text generation to code synthesis, depending on the capabilities of the model. If not explicitly provided, this defaults to the value set in `getOption("gptstudio.task")`.
custom_prompt	An optional parameter that provides a way to extend or customize the initial prompt with additional instructions or context.
process_response	A logical indicating whether to process the model's response. If `TRUE`, the response will be passed to `gptstudio_response_process()` for further processing. Defaults to `FALSE`. Refer to `gptstudio_response_process()` for more details.
...	Reserved for future use.

Value

Depending on the task and processing, the function returns the response from the model, which could be text, code, or any other structured output defined by the task and model capabilities. The precise format and content of the output depend on the specified options and the capabilities of the selected model.

Examples

```
## Not run:
```

```
# Basic usage with a text prompt:
result <- chat("What is the weather like today?")

# Advanced usage with custom settings, assuming appropriate global options are set:
result <- chat(
  prompt = "Write a simple function in R",
  skill = "advanced",
  style = "tidyverse",
  task = "coding"
)

# Usage with explicit service and model specification:
result <- chat(
  prompt = "Explain the concept of tidy data in R",
  service = "openai",
  model = "gpt-4-turbo-preview",
  skill = "intermediate",
  task = "general"
)

## End(Not run)
```

`chat_create_system_prompt` *Create system prompt*

Description

This function creates a customizable system prompt based on user-defined parameters such as coding style, skill level, and task. It supports customization for specific use cases through a custom prompt option.

Usage

```
chat_create_system_prompt(
  style = getOption("gptstudio.code_style"),
  skill = getOption("gptstudio.skill"),
  task = getOption("gptstudio.task"),
  custom_prompt = getOption("gptstudio.custom_prompt"),
  in_source = FALSE
)
```

Arguments

`style`	A character string indicating the preferred coding style. Valid values are "tidyverse", "base", "no preference". Defaults to `getOption(gptstudio.code_style)`.
`skill`	The self-described skill level of the programmer. Valid values are "beginner", "intermediate", "advanced", "genius". Defaults to `getOption(gptstudio.skill)`.

task	The specific task to be performed: "coding", "general", "advanced developer", or "custom". This influences the generated system prompt. Defaults to "coding".
custom_prompt	An optional custom prompt string to be utilized when task is set to "custom". Default is NULL.
in_source	A logical indicating whether the instructions are intended for use in a source script. This parameter is required and must be explicitly set to TRUE or FALSE. Default is FALSE.

Value

Returns a character string that forms a system prompt tailored to the specified parameters. The string provides guidance or instructions based on the user's coding style, skill level, and task.

Examples

```
## Not run:
chat_create_system_prompt(in_source = TRUE)
chat_create_system_prompt(
  style = "tidyverse",
  skill = "advanced",
  task = "coding",
  in_source = FALSE
)

## End(Not run)
```

---

chat_history_append *Append to chat history*

Description

This appends a new response to the chat history

Usage

```
chat_history_append(history, role, content, name = NULL)
```

Arguments

history	List containing previous responses.
role	Author of the message. One of c("user", "assistant")
content	Content of the message. If it is from the user most probably comes from an interactive input.
name	Name for the author of the message. Currently used to support rendering of help pages

Value

list of chat messages

chat_message_default *Default chat message*

Description

Default chat message

Usage

```
chat_message_default(translator = create_translator())
```

Arguments

translator	A Translator from shiny.i18n::Translator

Value

A default chat message for welcoming users.

check_api_connection_openai *Check API Connection*

Description

This generic function checks the API connection for a specified service by dispatching to related methods.

Usage

```
check_api_connection_openai(service, api_key)
```

Arguments

service	The name of the API service for which the connection is being checked.
api_key	The API key used for authentication.

Value

A logical value indicating whether the connection was successful.

create_chat_app_theme *Chat App Theme*

Description

Create a bslib theme that matches the user's RStudio IDE theme.

Usage

```
create_chat_app_theme(ide_colors = get_ide_theme_info())
```

Arguments

ide_colors	List containing the colors of the IDE theme.

Value

A bslib theme

create_chat_cohere *Create a chat with the Cohere Chat API*

Description

This function submits a user message to the Cohere Chat API, potentially along with other parameters such as chat history or connectors, and returns the API's response.

Usage

```
create_chat_cohere(
  prompt,
  chat_history = NULL,
  connectors = NULL,
  model = "command",
  api_key = Sys.getenv("COHERE_API_KEY")
)
```

Arguments

prompt	A string containing the user message.
chat_history	A list of previous messages for context, if any.
connectors	A list of connector objects, if any.
model	A string representing the Cohere model to be used, defaulting to "command". Other options include "command-light", "command-nightly", and "command-light-nightly".
api_key	The API key for accessing the Cohere API, defaults to the COHERE_API_KEY environment variable.

Value

The response from the Cohere Chat API containing the model's reply.

`create_completion_anthropic` *Generate text completions using Anthropic's API*

Description

Generate text completions using Anthropic's API

Usage

```
create_completion_anthropic(
  prompt = list(list(role = "user", content = "Hello")),
  system = NULL,
  model = "claude-3-haiku-20240307",
  max_tokens = 1028,
  key = Sys.getenv("ANTHROPIC_API_KEY")
)
```

Arguments

`prompt`	The prompt for generating completions
`system`	A system messages to instruct the model. Defaults to NULL.
`model`	The model to use for generating text. By default, the function will try to use "claude-2.1".
`max_tokens`	The maximum number of tokens to generate. Defaults to 256.
`key`	The API key for accessing Anthropic's API. By default, the function will try to use the `ANTHROPIC_API_KEY` environment variable.

Value

A list with the generated completions and other information returned by the API.

Examples

```
## Not run:
create_completion_anthropic(
  prompt = "\n\nHuman: Hello, world!\n\nAssistant:",
  model = "claude-3-haiku-20240307",
  max_tokens = 1028
)

## End(Not run)
```

`create_completion_azure_openai` *Generate text using Azure OpenAI's API*

Description

Use this function to generate text completions using OpenAI's API.

Usage

```
create_completion_azure_openai(
  prompt,
  task = Sys.getenv("AZURE_OPENAI_TASK"),
  base_url = Sys.getenv("AZURE_OPENAI_ENDPOINT"),
  deployment_name = Sys.getenv("AZURE_OPENAI_DEPLOYMENT_NAME"),
  token = Sys.getenv("AZURE_OPENAI_KEY"),
  api_version = Sys.getenv("AZURE_OPENAI_API_VERSION")
)
```

Arguments

`prompt`	a list to use as the prompt for generating completions
`task`	a character string for the API task (e.g. "completions"). Defaults to the Azure OpenAI task from environment variables if not specified.
`base_url`	a character string for the base url. It defaults to the Azure OpenAI endpoint from environment variables if not specified.
`deployment_name`	a character string for the deployment name. It will default to the Azure OpenAI deployment name from environment variables if not specified.
`token`	a character string for the API key. It will default to the Azure OpenAI API key from your environment variables if not specified.
`api_version`	a character string for the API version. It will default to the Azure OpenAI API version from your environment variables if not specified.

Value

a list with the generated completions and other information returned by the API

`create_completion_google` *Generate text completions using Google AI Studio's API*

Description

Generate text completions using Google AI Studio's API

Usage

```
create_completion_google(
  prompt,
  model = "gemini-pro",
  key = Sys.getenv("GOOGLE_API_KEY")
)
```

Arguments

`prompt`	The prompt for generating completions
`model`	The model to use for generating text. By default, the function will try to use "text-bison-001"
`key`	The API key for accessing Google AI Studio's API. By default, the function will try to use the `GOOGLE_API_KEY` environment variable.

Value

A list with the generated completions and other information returned by the API.

Examples

```
## Not run:
create_completion_google(
  prompt = "Write a story about a magic backpack",
  temperature = 1.0,
  candidate_count = 3
)

## End(Not run)
```

`create_completion_huggingface` *Generate text completions using HuggingFace's API*

Description

Generate text completions using HuggingFace's API

Usage

```
create_completion_huggingface(
  prompt,
  history = NULL,
  model = "tiiuae/falcon-7b-instruct",
  token = Sys.getenv("HF_API_KEY"),
  max_new_tokens = 250
)
```

Arguments

`prompt`	The prompt for generating completions
`history`	A list of the previous chat responses
`model`	The model to use for generating text
`token`	The API key for accessing HuggingFace's API. By default, the function will try to use the `HF_API_KEY` environment variable.
`max_new_tokens`	Maximum number of tokens to generate, defaults to 250

Value

A list with the generated completions and other information returned by the API.

Examples

```
## Not run:
create_completion_huggingface(
  model = "gpt2",
  prompt = "Hello world!"
)

## End(Not run)
```

create_completion_perplexity

Create a chat completion request to the Perplexity API

Description

This function sends a series of messages alongside a chosen model to the Perplexity API to generate a chat completion. It returns the API's generated responses.

Usage

```
create_completion_perplexity(
  prompt,
  model = "mistral-7b-instruct",
  api_key = Sys.getenv("PERPLEXITY_API_KEY")
)
```

Arguments

prompt	A list containing prompts to be sent in the chat.
model	A character string representing the Perplexity model to be used. Defaults to "mistral-7b-instruct".
api_key	The API key for accessing the Perplexity API. Defaults to the PERPLEXITY_API_KEY environment variable.

Value

The response from the Perplexity API containing the completion for the chat.

create_ide_matching_colors

Chat message colors in RStudio

Description

This returns a list of color properties for a chat message

Usage

```
create_ide_matching_colors(role, ide_colors = get_ide_theme_info())
```

Arguments

role	The role of the message author
ide_colors	List containing the colors of the IDE theme.

Value

list

`create_tmp_job_script` *Create a temporary job script*

Description

This function creates a temporary R script file that runs the Shiny application from the specified directory with the specified port and host.

Usage

```
create_tmp_job_script(appDir, port, host)
```

Arguments

`appDir`	The application to run. Should be one of the following: • A directory containing `server.R`, plus, either `ui.R` or a `www` directory that contains the file `index.html`. • A directory containing `app.R`. • An `.R` file containing a Shiny application, ending with an expression that produces a Shiny app object. • A list with `ui` and `server` components. • A Shiny app object created by `shinyApp()`.
`port`	The TCP port that the application should listen on. If the `port` is not specified, and the `shiny.port` option is set (with `options(shiny.port = XX)`), then that port will be used. Otherwise, use a random port between 3000:8000, excluding ports that are blocked by Google Chrome for being considered unsafe: 3659, 4045, 5060, 5061, 6000, 6566, 6665:6669 and 6697. Up to twenty random ports will be tried.
`host`	The IPv4 address that the application should listen on. Defaults to the `shiny.host` option, if set, or `"127.0.0.1"` if not. See Details.

Value

A string containing the path of a temporary job script

create_translator *Internationalization for the ChatGPT addin*

Description

The language can be set via `options("gptstudio.language" = "<language>")` (defaults to "en").

Usage

```
create_translator(language = getOption("gptstudio.language"))
```

Arguments

language	The language to be found in the translation JSON file.

Value

A Translator from `shiny.i18n::Translator`

get_available_endpoints *List supported endpoints*

Description

Get a list of the endpoints supported by gptstudio.

Usage

```
get_available_endpoints()
```

Value

A character vector

Examples

```
get_available_endpoints()
```

get_available_models *List supported models*

Description

Get a list of the models supported by the OpenAI API.

Usage

```
get_available_models(service)
```

Arguments

service	The API service

Value

A character vector

Examples

```
## Not run:
get_available_models()

## End(Not run)
```

get_ide_theme_info *Get IDE theme information.*

Description

This function returns a list with the current IDE theme's information.

Usage

```
get_ide_theme_info()
```

Value

A list with three components:

is_dark	A boolean indicating whether the current IDE theme is dark.
bg	The current IDE theme's background color.
fg	The current IDE theme's foreground color.

gptstudio_chat *Run Chat GPT Run the Chat GPT Shiny App as a background job and show it in the viewer pane*

Description

Run Chat GPT Run the Chat GPT Shiny App as a background job and show it in the viewer pane

Usage

```
gptstudio_chat(host = getOption("shiny.host", "127.0.0.1"))
```

Arguments

host	The IPv4 address that the application should listen on. Defaults to the `shiny.host` option, if set, or `"127.0.0.1"` if not. See Details.

Value

This function has no return value.

Examples

```
# Call the function as an RStudio addin
## Not run:
gptstudio_chat()

## End(Not run)
```

gptstudio_chat_in_source_addin *ChatGPT in Source*

Description

Call this function as a Rstudio addin to ask GPT to improve spelling and grammar of selected text.

Usage

```
gptstudio_chat_in_source_addin()
```

Value

This function has no return value.

Examples

```
# Select some text in a source file
# Then call the function as an RStudio addin
## Not run:
gptstudio_chat_in_source()

## End(Not run)
```

`gptstudio_comment_code` *Comment Code Addin*

Description

Call this function as a Rstudio addin to ask GPT to add comments to your code

Usage

```
gptstudio_comment_code()
```

Value

This function has no return value.

Examples

```
# Open a R file in Rstudio
# Then call the function as an RStudio addin
## Not run:
gptstudio_comment_code()

## End(Not run)
```

`gptstudio_create_skeleton` *Create a Request Skeleton*

Description

This function dynamically creates a request skeleton for different AI text generation services.

Usage

```
gptstudio_create_skeleton(
  service = "openai",
  prompt = "Name the top 5 packages in R.",
  history = list(list(role = "system", content = "You are an R chat assistant")),
  stream = TRUE,
  model = "gpt-3.5-turbo",
  ...
)
```

Arguments

service	The text generation service to use. Currently supports "openai", "huggingface", "anthropic", "google", "azure_openai", "ollama", and "perplexity".
prompt	The initial prompt or question to pass to the text generation service.
history	A list indicating the conversation history, where each element is a list with elements "role" (who is speaking; e.g., "system", "user") and "content" (what was said).
stream	Logical; indicates if streaming responses should be used. Currently, this option is not supported across all services.
model	The specific model to use for generating responses. Defaults to "gpt-3.5-turbo".
...	Additional arguments passed to the service-specific skeleton creation function.

Value

Depending on the selected service, returns a list that represents the configured request ready to be passed to the corresponding API.

Examples

```
## Not run:
request_skeleton <- gptstudio_create_skeleton(
  service = "openai",
  prompt = "Name the top 5 packages in R.",
  history = list(list(role = "system", content = "You are an R assistant")),
  stream = TRUE,
  model = "gpt-3.5-turbo"
)

## End(Not run)
```

gptstudio_request_perform *Perform API Request*

Description

This function provides a generic interface for calling different APIs (e.g., OpenAI, HuggingFace, Google AI Studio). It dispatches the actual API calls to the relevant method based on the `class` of the `skeleton` argument.

Usage

```
gptstudio_request_perform(skeleton, ...)
```

Arguments

`skeleton`	A `gptstudio_request_skeleton` object
`...`	Extra arguments (e.g., `stream_handler`)

Value

A `gptstudio_response_skeleton` object

Examples

```
## Not run:
gptstudio_request_perform(gptstudio_skeleton)

## End(Not run)
```

gptstudio_response_process *Call API*

Description

This function provides a generic interface for calling different APIs (e.g., OpenAI, HuggingFace, Google AI Studio). It dispatches the actual API calls to the relevant method based on the `class` of the `skeleton` argument.

Usage

```
gptstudio_response_process(skeleton, ...)
```

Arguments

skeleton	A gptstudio_response_skeleton object
...	Extra arguments, not currently used

Value

A gptstudio_request_skeleton with updated history and prompt removed

Examples

```
## Not run:
gptstudio_response_process(gptstudio_skeleton)

## End(Not run)
```

---

gptstudio_sitrep *Current Configuration for gptstudio*

Description

This function prints out the current configuration settings for gptstudio and checks API connections if verbose is TRUE.

Usage

```
gptstudio_sitrep(verbose = TRUE)
```

Arguments

verbose	Logical value indicating whether to output additional information, such as API connection checks. Defaults to TRUE.

Value

Invisibly returns NULL, as the primary purpose of this function is to print to the console.

Examples

```
gptstudio_sitrep(verbose = FALSE) # Print basic settings, no API checks
gptstudio_sitrep() # Print settings and check API connections
```

gptstudio_skeleton_build *Construct a GPT Studio request skeleton.*

Description

Construct a GPT Studio request skeleton.

Usage

```
gptstudio_skeleton_build(skeleton, skill, style, task, custom_prompt, ...)
```

Arguments

skeleton	A GPT Studio request skeleton object.
skill	The skill level of the user for the chat conversation. This can be set through the "gptstudio.skill" option. Default is the "gptstudio.skill" option. Options are "beginner", "intermediate", "advanced", and "genius".
style	The style of code to use. Applicable styles can be retrieved from the "gptstudio.code_style" option. Default is the "gptstudio.code_style" option. Options are "base", "tidyverse", or "no preference".
task	Specifies the task that the assistant will help with. Default is "coding". Others are "general", "advanced developer", and "custom".
custom_prompt	This is a custom prompt that may be used to guide the AI in its responses. Default is NULL. It will be the only content provided to the system prompt.
...	Additional arguments.

Value

An updated GPT Studio request skeleton.

gptstudio_spelling_grammar *Spelling and Grammar Addin*

Description

Call this function as a Rstudio addin to ask GPT to improve spelling and grammar of selected text.

Usage

```
gptstudio_spelling_grammar()
```

Value

This function has no return value.

Examples

```
# Select some text in Rstudio
# Then call the function as an RStudio addin
## Not run:
gptstudio_spelling_grammar()

## End(Not run)
```

`mod_app_server` *App Server*

Description

App Server

Usage

```
mod_app_server(id, ide_colors = get_ide_theme_info())
```

Arguments

`id`	id of the module
`ide_colors`	List containing the colors of the IDE theme.

`mod_app_ui` *App UI*

Description

App UI

Usage

```
mod_app_ui(id, ide_colors = get_ide_theme_info())
```

Arguments

`id`	id of the module
`ide_colors`	List containing the colors of the IDE theme.

mod_chat_server *Chat server*

Description

Chat server

Usage

```
mod_chat_server(
  id,
  ide_colors = get_ide_theme_info(),
  translator = create_translator(),
  settings,
  history
)
```

Arguments

id	id of the module
ide_colors	List containing the colors of the IDE theme.
translator	Translator from shiny.i18n::Translator
settings, history	Reactive values from the settings and history module

mod_chat_ui *Chat UI*

Description

Chat UI

Usage

```
mod_chat_ui(id, translator = create_translator())
```

Arguments

id	id of the module
translator	A Translator from shiny.i18n::Translator

`OpenaiStreamParser` *Stream handler for chat completions*

Description

Stream handler for chat completions

Stream handler for chat completions

Details

R6 class that allows to handle chat completions chunk by chunk. It also adds methods to retrieve relevant data. This class DOES NOT make the request.

Because `curl::curl_fetch_stream` blocks the R console until the stream finishes, this class can take a shiny session object to handle communication with JS without recurring to a `shiny::observe` inside a module server.

Super class

`SSEparser::SSEparser` -> `OpenaiStreamParser`

Public fields

`shinySession` Holds the `session` provided at initialization

`user_prompt` The `user_prompt` provided at initialization, after being formatted with markdown.

`value` The content of the stream. It updates constantly until the stream ends.

Methods

Public methods:

- `OpenaiStreamParser$new()`
- `OpenaiStreamParser$append_parsed_sse()`
- `OpenaiStreamParser$clone()`

Method `new()`**:** Start a StreamHandler. Recommended to be assigned to the `stream_handler` name.

Usage:

```
OpenaiStreamParser$new(session = NULL, user_prompt = NULL)
```

Arguments:

`session` The shiny session it will send the message to (optional).

`user_prompt` The prompt for the chat completion. Only to be displayed in an HTML tag containing the prompt. (Optional).

Method `append_parsed_sse()`**:** Overwrites `SSEparser$append_parsed_sse()` to be able to send a custom message to a shiny session, escaping shiny's reactivity.

Usage:

```
OpenaiStreamParser$append_parsed_sse(parsed_event)
```

Arguments:

`parsed_event` An already parsed server-sent event to append to the events field.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
OpenaiStreamParser$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`openai_create_chat_completion` *Generate text completions using OpenAI's API for Chat*

Description

Generate text completions using OpenAI's API for Chat

Usage

```
openai_create_chat_completion(
  prompt = "<|endoftext|>",
  model = getOption("gptstudio.model"),
  openai_api_key = Sys.getenv("OPENAI_API_KEY"),
  task = "chat/completions"
)
```

Arguments

`prompt`	The prompt for generating completions
`model`	The model to use for generating text
`openai_api_key`	The API key for accessing OpenAI's API. By default, the function will try to use the `OPENAI_API_KEY` environment variable.
`task`	The task that specifies the API url to use, defaults to "completions" and "chat/completions" is required for ChatGPT model.

Value

A list with the generated completions and other information returned by the API.

Examples

```
## Not run:
openai_create_completion(
  model = "text-davinci-002",
  prompt = "Hello world!"
)

## End(Not run)
```

---

open_bg_shinyapp *Open browser to local Shiny app*

Description

This function takes in the host and port of a local Shiny app and opens the app in the default browser.

Usage

```
open_bg_shinyapp(host, port)
```

Arguments

host	A character string representing the IP address or domain name of the server where the Shiny app is hosted.
port	An integer representing the port number on which the Shiny app is hosted.

Value

None (opens the Shiny app in the viewer pane or browser window)

---

prepare_chat_history *Prepare chat completion prompt*

Description

This function prepares the chat completion prompt to be sent to the OpenAI API. It also generates a system message according to the given parameters and inserts it at the beginning of the conversation.

Usage

```
prepare_chat_history(
  history = NULL,
  style = getOption("gptstudio.code_style"),
  skill = getOption("gptstudio.skill"),
  task = "coding",
  custom_prompt = NULL
)
```

Arguments

history	A list of previous messages in the conversation. This can include roles such as 'system', 'user', or 'assistant'. System messages are discarded. Default is NULL.i
style	The style of code to use. Applicable styles can be retrieved from the "gptstudio.code_style" option. Default is the "gptstudio.code_style" option. Options are "base", "tidyverse", or "no preference".
skill	The skill level of the user for the chat conversation. This can be set through the "gptstudio.skill" option. Default is the "gptstudio.skill" option. Options are "beginner", "intermediate", "advanced", and "genius".
task	Specifies the task that the assistant will help with. Default is "coding". Others are "general", "advanced developer", and "custom".
custom_prompt	This is a custom prompt that may be used to guide the AI in its responses. Default is NULL. It will be the only content provided to the system prompt.

Value

A list where the first entry is an initial system message followed by any non-system entries from the chat history.

---

query_api_anthropic *A function that sends a request to the Anthropic API and returns the response.*

Description

A function that sends a request to the Anthropic API and returns the response.

Usage

```
query_api_anthropic(request_body, key = Sys.getenv("ANTHROPIC_API_KEY"))
```

Arguments

request_body	A list that contains the parameters for the task.
key	String containing an Anthropic API key. Defaults to the ANTHROPIC_API_KEY environmental variable if not specified.

Value

The response from the API.

query_api_cohere *Send a request to the Cohere Chat API and return the response*

Description

This function sends a JSON post request to the Cohere Chat API, retries on failure up to three times, and returns the response. The function handles errors by providing a descriptive message and failing gracefully.

Usage

```
query_api_cohere(request_body, api_key = Sys.getenv("COHERE_API_KEY"))
```

Arguments

request_body	A list containing the body of the POST request.
api_key	String containing a Cohere API key. Defaults to the COHERE_API_KEY environmental variable if not specified.

Value

A parsed JSON object as the API response.

query_api_google *A function that sends a request to the Google AI Studio API and returns the response.*

Description

A function that sends a request to the Google AI Studio API and returns the response.

Usage

```
query_api_google(model, request_body, key = Sys.getenv("GOOGLE_API_KEY"))
```

Arguments

model	A character string that specifies the model to send to the API.
request_body	A list that contains the parameters for the task.
key	String containing a Google AI Studio API key. Defaults to the GOOGLE_API_KEY environmental variable if not specified.

Value

The response from the API.

query_api_huggingface *A function that sends a request to the HuggingFace API and returns the response.*

Description

A function that sends a request to the HuggingFace API and returns the response.

Usage

```
query_api_huggingface(task, request_body, token = Sys.getenv("HF_API_KEY"))
```

Arguments

task	A character string that specifies the task to send to the API.
request_body	A list that contains the parameters for the task.
token	String containing a HuggingFace API key. Defaults to the HF_API_KEY environmental variable if not specified.

Value

The response from the API.

query_api_perplexity *Send a request to the Perplexity API and return the response*

Description

This function sends a JSON post request to the Perplexity API, retries on failure up to three times, and returns the response. The function handles errors by providing a descriptive message and failing gracefully.

Usage

```
query_api_perplexity(request_body, api_key = Sys.getenv("PERPLEXITY_API_KEY"))
```

Arguments

request_body	A list containing the body of the POST request.
api_key	String containing a Perplexity API key. Defaults to the PERPLEXITY_API_KEY environmental variable if not specified.

Value

A parsed JSON object as the API response.

query_openai_api *A function that sends a request to the OpenAI API and returns the response.*

Description

A function that sends a request to the OpenAI API and returns the response.

Usage

```
query_openai_api(
  task,
  request_body,
  openai_api_key = Sys.getenv("OPENAI_API_KEY")
)
```

Arguments

task	A character string that specifies the task to send to the API.
request_body	A list that contains the parameters for the task.
openai_api_key	String containing an OpenAI API key. Defaults to the OPENAI_API_KEY environmental variable if not specified.

Value

The response from the API.

random_port *Generate a random safe port number*

Description

This function generates a random port allowed by shiny::runApp.

Usage

```
random_port()
```

Value

A single integer representing the randomly selected safe port number.

request_base *Base for a request to the OPENAI API*

Description

This function sends a request to a specific OpenAI API `task` endpoint at the base URL `https://api.openai.com/v1`, and authenticates with an API key using a Bearer token.

Usage

```
request_base(task, token = Sys.getenv("OPENAI_API_KEY"))
```

Arguments

`task`	character string specifying an OpenAI API endpoint task
`token`	String containing an OpenAI API key. Defaults to the OPENAI_API_KEY environmental variable if not specified.

Value

An httr2 request object

request_base_anthropic *Base for a request to the Anthropic API*

Description

This function sends a request to the Anthropic API endpoint and authenticates with an API key.

Usage

```
request_base_anthropic(key = Sys.getenv("ANTHROPIC_API_KEY"))
```

Arguments

`key`	String containing an Anthropic API key. Defaults to the ANTHROPIC_API_KEY environmental variable if not specified.

Value

An httr2 request object

request_base_cohere *Base for a request to the Cohere Chat API*

Description

This function sets up a POST request to the Cohere Chat API's chat endpoint and includes necessary headers such as 'accept', 'content-type', and 'Authorization' with a bearer token.

Usage

```
request_base_cohere(api_key = Sys.getenv("COHERE_API_KEY"))
```

Arguments

api_key	String containing a Cohere API key. Defaults to the COHERE_API_KEY environment variable if not specified.

Value

An `httr2` request object pre-configured with the API endpoint and required headers.

request_base_google *Base for a request to the Google AI Studio API*

Description

This function sends a request to a specific Google AI Studio API endpoint and authenticates with an API key.

Usage

```
request_base_google(model, key = Sys.getenv("GOOGLE_API_KEY"))
```

Arguments

model	character string specifying a Google AI Studio API model
key	String containing a Google AI Studio API key. Defaults to the GOOGLE_API_KEY environmental variable if not specified.

Value

An httr2 request object

request_base_huggingface *Base for a request to the HuggingFace API*

Description

This function sends a request to a specific HuggingFace API endpoint and authenticates with an API key using a Bearer token.

Usage

```
request_base_huggingface(task, token = Sys.getenv("HF_API_KEY"))
```

Arguments

task	character string specifying a HuggingFace API endpoint task
token	String containing a HuggingFace API key. Defaults to the HF_API_KEY environmental variable if not specified.

Value

An httr2 request object

request_base_perplexity *Base for a request to the Perplexity API*

Description

This function sets up a POST request to the Perplexity API's chat/completions endpoint and includes necessary headers such as 'accept', 'content-type', and 'Authorization' with a bearer token.

Usage

```
request_base_perplexity(api_key = Sys.getenv("PERPLEXITY_API_KEY"))
```

Arguments

api_key	String containing a Perplexity API key. Defaults to the PERPLEXITY_API_KEY environment variable if not specified.

Value

An `httr2` request object pre-configured with the API endpoint and required headers.

rgb_str_to_hex *RGB str to hex*

Description

RGB str to hex

Usage

```
rgb_str_to_hex(rgb_string)
```

Arguments

rgb_string	The RGB string as returned by `rstudioapi::getThemeInfo()`

Value

hex color

run_app_as_bg_job *Run an R Shiny app in the background*

Description

This function runs an R Shiny app as a background job using the specified directory, name, host, and port.

Usage

```
run_app_as_bg_job(appDir = ".", job_name, host, port)
```

Arguments

appDir	The application to run. Should be one of the following: • A directory containing `server.R`, plus, either `ui.R` or a `www` directory that contains the file `index.html`. • A directory containing `app.R`. • An `.R` file containing a Shiny application, ending with an expression that produces a Shiny app object. • A list with `ui` and `server` components. • A Shiny app object created by `shinyApp()`.
job_name	The name of the background job to be created
host	The IPv4 address that the application should listen on. Defaults to the `shiny.host` option, if set, or `"127.0.0.1"` if not. See Details.

port	The TCP port that the application should listen on. If the port is not specified, and the shiny.port option is set (with options(shiny.port = XX)), then that port will be used. Otherwise, use a random port between 3000:8000, excluding ports that are blocked by Google Chrome for being considered unsafe: 3659, 4045, 5060, 5061, 6000, 6566, 6665:6669 and 6697. Up to twenty random ports will be tried.

Value

This function returns nothing because is meant to run an app as a side effect.

`run_chatgpt_app` *Run the ChatGPT app*

Description

This starts the chatgpt app. It is exported to be able to run it from an R script.

Usage

```
run_chatgpt_app(
  ide_colors = get_ide_theme_info(),
  host = getOption("shiny.host", "127.0.0.1"),
  port = getOption("shiny.port")
)
```

Arguments

ide_colors	List containing the colors of the IDE theme.
host	The IPv4 address that the application should listen on. Defaults to the shiny.host option, if set, or "127.0.0.1" if not. See Details.
port	The TCP port that the application should listen on. If the port is not specified, and the shiny.port option is set (with options(shiny.port = XX)), then that port will be used. Otherwise, use a random port between 3000:8000, excluding ports that are blocked by Google Chrome for being considered unsafe: 3659, 4045, 5060, 5061, 6000, 6566, 6665:6669 and 6697. Up to twenty random ports will be tried.

Value

Nothing.

`streamingMessage` *Streaming message*

Description

Places an invisible empty chat message that will hold a streaming message. It can be reset dynamically inside a shiny app

Usage

```
streamingMessage(
  ide_colors = get_ide_theme_info(),
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

`ide_colors`	List containing the colors of the IDE theme.
`width, height`	Must be a valid CSS unit (like `'100%'`, `'400px'`, `'auto'`) or a number, which will be coerced to a string and have `'px'` appended.
`element_id`	The element's id

`streamingMessage-shiny`

Shiny bindings for streamingMessage

Description

Output and render functions for using streamingMessage within Shiny applications and interactive Rmd documents.

Usage

```
streamingMessageOutput(outputId, width = "100%", height = NULL)

renderStreamingMessage(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

outputId	output variable to read from
width, height	Must be a valid CSS unit (like '100%', '400px', 'auto') or a number, which will be coerced to a string and have 'px' appended.
expr	An expression that generates a streamingMessage
env	The environment in which to evaluate expr.
quoted	Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable.

---

stream_chat_completion

Stream Chat Completion

Description

stream_chat_completion sends the prepared chat completion request to the OpenAI API and retrieves the streamed response.

Usage

```
stream_chat_completion(
  messages = NULL,
  element_callback = cat,
  model = "gpt-3.5-turbo",
  openai_api_key = Sys.getenv("OPENAI_API_KEY")
)
```

Arguments

messages	A list of messages in the conversation, including the current user prompt (optional).
element_callback	A callback function to handle each element of the streamed response (optional).
model	A character string specifying the model to use for chat completion. The default model is "gpt-3.5-turbo".
openai_api_key	A character string of the OpenAI API key. By default, it is fetched from the "OPENAI_API_KEY" environment variable. Please note that the OpenAI API key is sensitive information and should be treated accordingly.

Value

The same as curl::curl_fetch_stream

style_chat_history *Style Chat History*

Description

This function processes the chat history, filters out system messages, and formats the remaining messages with appropriate styling.

Usage

```
style_chat_history(history, ide_colors = get_ide_theme_info())
```

Arguments

history	A list of chat messages with elements containing 'role' and 'content'.
ide_colors	List containing the colors of the IDE theme.

Value

A list of formatted chat messages with styling applied, excluding system messages.

Examples

```
chat_history_example <- list(
  list(role = "user", content = "Hello, World!"),
  list(role = "system", content = "System message"),
  list(role = "assistant", content = "Hi, how can I help?")
)

## Not run:
style_chat_history(chat_history_example)

## End(Not run)
```

style_chat_message *Style chat message*

Description

Style a message based on the role of its author.

Usage

```
style_chat_message(message, ide_colors = get_ide_theme_info())
```

Arguments

message	A chat message.
ide_colors	List containing the colors of the IDE theme.

Value

An HTML element.

`text_area_input_wrapper` *Custom textAreaInput*

Description

Modified version of `textAreaInput()` that removes the label container. It's used in `mod_prompt_ui()`

Usage

```
text_area_input_wrapper(
  inputId,
  label,
  value = "",
  width = NULL,
  height = NULL,
  cols = NULL,
  rows = NULL,
  placeholder = NULL,
  resize = NULL,
  textarea_class = NULL
)
```

Arguments

inputId	The `input` slot that will be used to access the value.
label	Display label for the control, or `NULL` for no label.
value	Initial value.
width	The width of the input, e.g. `'400px'`, or `'100%'`; see `validateCssUnit()`.
height	The height of the input, e.g. `'400px'`, or `'100%'`; see `validateCssUnit()`.
cols	Value of the visible character columns of the input, e.g. `80`. This argument will only take effect if there is not a CSS `width` rule defined for this element; such a rule could come from the `width` argument of this function or from a containing page layout such as `fluidPage()`.
rows	The value of the visible character rows of the input, e.g. `6`. If the `height` argument is specified, `height` will take precedence in the browser's rendering.

placeholder	A character string giving the user a hint as to what can be entered into the control. Internet Explorer 8 and 9 do not support this option.
resize	Which directions the textarea box can be resized. Can be one of "both", "none", "vertical", and "horizontal". The default, NULL, will use the client browser's default setting for resizing textareas.
textarea_class	Class to be applied to the textarea element

Value

A modified textAreaInput

---

welcomeMessage *Welcome message*

Description

HTML widget for showing a welcome message in the chat app. This has been created to be able to bind the message to a shiny event to trigger a new render.

Usage

```
welcomeMessage(
  ide_colors = get_ide_theme_info(),
  translator = create_translator(),
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

ide_colors	List containing the colors of the IDE theme.
translator	A Translator from shiny.i18n::Translator
width, height	Must be a valid CSS unit (like '100%', '400px', 'auto') or a number, which will be coerced to a string and have 'px' appended.
element_id	The element's id

`welcomeMessage-shiny` *Shiny bindings for welcomeMessage*

Description

Output and render functions for using welcomeMessage within Shiny applications and interactive Rmd documents.

Usage

```
welcomeMessageOutput(outputId, width = "100%", height = NULL)

renderWelcomeMessage(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

`outputId`	output variable to read from
`width, height`	Must be a valid CSS unit (like `'100%'`, `'400px'`, `'auto'`) or a number, which will be coerced to a string and have `'px'` appended.
`expr`	An expression that generates a welcomeMessage
`env`	The environment in which to evaluate `expr`.
`quoted`	Is `expr` a quoted expression (with `quote()`)? This is useful if you want to save an expression in a variable.

Index