Package 'gptstudio'

July 11, 2023

Type Package

Title Use Large Language Models Directly in your Development Environment

Version 0.3.0

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

Imports assertthat, bslib (>= 0.4.2), callr, cli, colorspace, glue, grDevices, htmltools, htmlwidgets, httr2, jsonlite, magrittr, purrr, rlang, rstudioapi (>= 0.12), shiny, shiny.i18n, stringr (>= 1.5.0), utils, waiter, yaml

Suggests mockr, shinytest2, spelling, testthat (>= 3.0.0), uuid, withr

Depends R (>= 4.0)

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Language en-US

RoxygenNote 7.2.3

NeedsCompilation no

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

Repository CRAN

Date/Publication 2023-07-11 12:00:16 UTC

R topics documented:

`addin_chatgpt` *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

```
addin_chatgpt(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:
addin_chatgpt()

## End(Not run)
```

addin_chatgpt_in_source *ChatGPT in Source*

Description

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

Usage

```
addin_chatgpt_in_source()
```

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:
addin_chatgpt_in_source()

## End(Not run)
```

addin_comment_code *Comment Code Addin*

Description

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

Usage

```
addin_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:
addin_comment_code()

## End(Not run)
```

addin_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

```
addin_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:
addin_spelling_grammar()

## End(Not run)
```

chat_create_system_prompt *Create system prompt*

Description

This creates a system prompt based on the user defined parameters.

Usage

```
chat_create_system_prompt(
  style = c("tidyverse", "base", "no preference", NULL),
  skill = c("beginner", "intermediate", "advanced", "genius", NULL),
  task = c("coding", "general", "advanced developer", "custom"),
  custom_prompt = NULL,
  in_source
)
```

Arguments

style	A character string indicating the preferred coding style, the default is "tidyverse".
skill	The self-described skill level of the programmer, default is "beginner"
task	The task to be performed: "coding", "general", or "advanced developer".
custom_prompt	An optional custom prompt to be displayed.
in_source	Whether to add instructions to act as in a source script.

Value

A string

---

chat_history_append *Append to chat history*

Description

This appends a new response to the chat history

Usage

```
chat_history_append(history, role, content)
```

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.

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 *Check API setup*

Description

This function checks whether the API key provided in the OPENAI_API_KEY environment variable is valid. This function will not re-check an API if it has already been validated in the current session.

Usage

```
check_api()
```

Value

Nothing is returned. If the API key is valid, a success message is printed. If the API key is invalid, an error message is printed and the function aborts.

Examples

```
# Call the function to check the API key
## Not run:
check_api()

## End(Not run)
```

check_api_connection *Check connection to OpenAI's API works*

Description

This function checks whether the API key provided in the OPENAI_API_KEY environment variable is valid.

Usage

```
check_api_connection(api_key, update_api = TRUE, verbose = FALSE)
```

Arguments

api_key	An API key.
update_api	Whether to attempt to update api if invalid
verbose	Whether to provide information about the API connection

Value

Nothing is returned. If the API key is valid, a success message is printed. If the API key is invalid, an error message is printed and the function is aborted.

Examples

```
# Call the function with an API key
## Not run:
check_api_connection("my_api_key")

## End(Not run)
# Call the function with an API key and avoid updating the API key
## Not run:
check_api_connection("my_api_key", update_api = FALSE)

## End(Not run)
```

check_api_key *Check API key*

Description

This function checks whether the API key provided as an argument is in the correct format.

Usage

```
check_api_key(api_key, update_api = TRUE)
```

Arguments

api_key	An API key.
update_api	Whether to attempt to update api if invalid

Value

Nothing is returned. If the API key is in the correct format, a success message is printed. If the API key is not in the correct format, an error message is printed and the function aborts.

Examples

```
# Call the function with an API key
## Not run:
check_api_key("my_api_key")

## End(Not run)
# Call the function with an API key and avoid updating the API key
## Not run:
check_api_key("my_api_key", update_api = FALSE)

## End(Not run)
```

---

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

Description

Generate text completions using Anthropic's API

Usage

```
create_completion_anthropic(
  prompt,
  history = NULL,
  model = "claude-1",
  max_tokens_to_sample = 256,
  key = Sys.getenv("ANTHROPIC_API_KEY")
)
```

Arguments

`prompt`	The prompt for generating completions
`history`	A list of the previous chat responses
`model`	The model to use for generating text. By default, the function will try to use "claude-1".
`max_tokens_to_sample`	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-1",
  max_tokens_to_sample = 256
)

## 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. 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

Examples

```
## Not run:
create_completion_azure_openai(
  prompt = list(list(role = "user", content = "Hello world!"
)

## 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_palm *Generate text completions using PALM (MakerSuite)'s API*

Description

Generate text completions using PALM (MakerSuite)'s API

Usage

```
create_completion_palm(
  prompt,
  model = "text-bison-001",
  key = Sys.getenv("PALM_API_KEY"),
  temperature = 0.5,
  candidate_count = 1
)
```

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 PALM (MakerSuite)'s API. By default, the function will try to use the PALM_API_KEY environment variable.
temperature	The temperature to control the randomness of the model's output
candidate_count	The number of completion candidates to generate

Value

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

Examples

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

## End(Not run)
```

`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

```
get_available_endpoints()
```

---

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_job *Perform Job*

Description

Combined job to build the skeleton, perform the api request, and process the response

Usage

```
gptstudio_job(
  skeleton = gptstudio_create_skeleton(),
  skill = getOption("gptstudio.skill"),
  style = getOption("gptstudio.code_style"),
  task = getOption("gptstudio.task"),
  custom_prompt = getOption("gptstudio.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.

gptstudio_request_perform *Perform API Request*

Description

This function provides a generic interface for calling different APIs (e.g., OpenAI, HuggingFace, PALM (MakerSuite)). 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, PALM (MakerSuite)). 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_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.

gpt_chat *ChatGPT in RStudio*

Description

This function uses the ChatGPT API tailored to a user-provided style and skill level.

Usage

```
gpt_chat(
  history,
  style = getOption("gptstudio.code_style"),
  skill = getOption("gptstudio.skill"),
  model = getOption("gptstudio.model")
)
```

Arguments

history	A list of the previous chat responses
style	A character string indicating the preferred coding style, the default is "tidyverse".
skill	The self-described skill level of the programmer, default is "beginner"
model	The name of the GPT model to use.

Value

A list containing the instructions for answering the question, the context in which the question was asked, and the suggested answer.

Examples

```
## Not run:
# Example 1: Get help with a tidyverse question
tidyverse_query <- "How can I filter rows of a data frame?"
tidyverse_response <- gpt_chat(
  query = tidyverse_query,
  style = "tidyverse",
  skill = "beginner"
)
print(tidyverse_response)

# Example 2: Get help with a base R question
base_r_query <- "How can I merge two data frames?"
base_r_response <- gpt_chat(
  query = base_r_query,
  style = "base",
  skill = "intermediate"
)
print(base_r_response)

# Example 3: No style preference
no_preference_query <- "What is the best way to handle missing values in R?"
no_preference_response <- gpt_chat(
  query = no_preference_query,
  style = "no preference",
  skill = "advanced"
)
print(no_preference_response)

## End(Not run)
```

---

gpt_chat_in_source *ChatGPT in Source*

Description

Provides the same functionality as `gpt_chat()` with minor modifications to give more useful output in a source (i.e., *.R) file.

Usage

```
gpt_chat_in_source(
  history = NULL,
  task = NULL,
  style = getOption("gptstudio.code_style"),
  skill = getOption("gptstudio.skill")
)
```

Arguments

`history`	A list of the previous chat responses
`task`	Specific instructions to provide to the model as a system prompt
`style`	A character string indicating the preferred coding style, the default is "tidyverse".
`skill`	The self-described skill level of the programmer, default is "beginner"

Value

A list containing the instructions for answering the question, the context in which the question was asked, and the suggested answer.

Examples

```
## Not run:
# Example 1: Get help with a tidyverse question in a source file
# Select the following code comment in RStudio and run gpt_chat_in_source()
# How can I filter rows of a data frame?
tidyverse_response <- gpt_chat_in_source(
  style = "tidyverse",
  skill = "beginner"
)

# Example 2: Get help with a base R question in a source file
# Select the following code comment in RStudio and run gpt_chat_in_source()
# How can I merge two data frames?
base_r_response <- gpt_chat_in_source(style = "base", skill = "intermediate")

# Example 3: No style preference in a source file
# Select the following code comment in RStudio and run gpt_chat_in_source()
# What is the best way to handle missing values in R?
no_preference_response <- gpt_chat_in_source(
  style = "no preference",
  skill = "advanced"
)
```

```
## 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()
)
```

Arguments

id	id of the module
ide_colors	List containing the colors of the IDE theme.
translator	Translator from shiny.i18n::Translator

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

`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)
```

openai_stream_parse *OpenAI Stream Parse*

Description

This function handles the streaming data from the OpenAI API. It concatenates the raw data chunks, attempts to parse JSON and handles any error messages.

Usage

```
openai_stream_parse(x)
```

Arguments

x	A raw vector representing a chunk of data from the API stream.

Details

This function was inspired by the {chattr} R package (https://github.com/mlverse/chattr).

Value

If parsing is successful, a character string of the API response is returned. In case of an error, an error message is returned instead.

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_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_palm *A function that sends a request to the PALM (MakerSuite) API and returns the response.*

Description

A function that sends a request to the PALM (MakerSuite) API and returns the response.

Usage

```
query_api_palm(model, request_body, key = Sys.getenv("PALM_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 PALM (MakerSuite) API key. Defaults to the PALM_API_KEY environmental variable if not specified.

Value

The response from the API.

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_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_palm *Base for a request to the PALM (MakerSuite) API*

Description

This function sends a request to a specific PALM (MakerSuite) API endpoint and authenticates with an API key.

Usage

```
request_base_palm(model, key = Sys.getenv("PALM_API_KEY"))
```

Arguments

model	character string specifying a PALM (MakerSuite) API model
key	String containing a PALM (MakerSuite) API key. Defaults to the PALM_API_KEY environmental variable if not specified.

Value

An httr2 request object

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,
  elementId = 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.
elementId	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. The results are then stored in a temporary file.

Usage

```
stream_chat_completion(
  prompt,
  model = "gpt-3.5-turbo",
  openai_api_key = Sys.getenv("OPENAI_API_KEY")
)
```

Arguments

prompt	A list of messages. Each message is a list that includes a "role" and "content". The "role" can be "system", "user", or "assistant". The "content" is the text of the message from the role.
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

A character string specifying the path to the tempfile that contains the full response from the OpenAI API.

Examples

```
## Not run:
# Get API key from your environment variables
openai_api_key <- Sys.getenv("OPENAI_API_KEY")

# Define the prompt
prompt <- list(
  list(role = "system", content = "You are a helpful assistant."),
  list(role = "user", content = "Who won the world series in 2020?")
)

# Call the function
result <- stream_chat_completion(prompt = prompt, openai_api_key = openai_api_key)

# Print the result
print(result)

## End(Not run)
```

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,
  elementId = 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.
elementId	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