Package: gpttools (via r-universe)

November 19, 2024

Title Extensions and Tools for gptstudio

Version 0.0.8.9016

Description gpttools is an R package that provides extensions to gptstudio to provide devtools-like functionality using the latest natural language processing (NLP) models. It is designed to make package development easier by providing a range of tools and functions that can be used to improve the quality of your package's documentation, testing, and maybe even functionality.

License MIT + file LICENSE

URL https://github.com/JamesHWade/gpttools

BugReports https://github.com/JamesHWade/gpttools/issues

Depends R (>= 4.1)

Imports arrow, cli, dplyr, glue, gptstudio (>= 0.3.1), httr2, jsonlite, lsa, purrr (>= 1.0.0), readr, rlang (>= 0.4.11), rstudioapi (>= 0.12), rvest, shiny, stringr, tibble, tidyr, tokenizers, tools, urltools, xml2, yaml

Suggests AzureRMR, bsicons, bslib, covr, furrr, future, htmltools, httr, knitr, later, mockr, pak, pdftools, precommit, reprex, reticulate, rmarkdown, roxygen2, shinyjs, shinytest2, skimr, spelling, testthat (>= 3.0.0), tidymodels, tidyverse, tuneR, uuid, waiter, withr

VignetteBuilder knitr

Remotes michelnivard/gptstudio, rstudio/rstudioapi

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Config/pak/sysreqs cmake make libicu-dev libxml2-dev libssl-dev
libx11-dev zlib1g-dev

Repository https://jameshwade.r-universe.dev

RemoteUrl https://github.com/JamesHWade/gpttools

RemoteRef main

RemoteSha 508a179a62f424215e610528b66c902d3a3bb2ed

Contents

`addin_run_scrape_pkgs` *Addin to scrape installed packages*

Description

Invokes RStudio addin functionality to scrape select installed packages and create indices for use in the "Chat with Retrieval" application.

Usage

```
addin_run_scrape_pkgs()
```

Value

No return value, called for side effects only.

Note

This addin requires RStudio to be available and will stop with an error message if RStudio API is not accessible.

Examples

```
# This function is typically run within RStudio as an Addin.
# It would not be called directly in the code.
addin_scrape_pkgs()
```

`addin_run_select_pkgs` *Run Package Selector App*

Description

Run the package selector shiny app

Usage

```
addin_run_select_pkgs()
```

Value

This function has no return value.

`add_roxygen_addin` *Add Roxygen documentation to a function*

Description

This function uses the OpenAI API to generate a roxygen skeleton for the current selection in RStudio. The roxygen skeleton is then inserted into the document using prompt: "insert roxygen skeleton to document this function"

Usage

```
add_roxygen_addin()
```

Value

NULL (nothing is returned; the generated roxygen skeleton is inserted into the document).

chat *Chat for gpttools*

Description

This function provides a high-level interface for communicating with various services and models supported by gpttools. It orchestrates the creation, configuration, and execution of a request based on user inputs and options set for gpttools 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("gpttools.service"),
  history = list(list(role = "system", content = "You are an R chat assistant")),
  stream = getOption("gpttools.stream", TRUE),
  model = getOption("gpttools.model"),
  skill = getOption("gpttools.skill", NULL),
  style = getOption("gpttools.code_style", "no preference"),
  task = NULL,
  custom_prompt = NULL,
  process_response = FALSE,
  where = "console",
  ...
)
```

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.,
`where`	A character string indicating the location or environment where the chat is taking place. Options are `c("console", "source", and "shiny")`. The default is `""`, which means the chat is taking place in the R console.
`...`	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_with_context` *chat_with_context*

Description

This function allows you to chat with a chatbot that answers questions based on the provided context and chat history. It uses GPT-4 architecture to generate responses.

Usage

```
chat_with_context(
  query,
  service = "openai",
  model = "gpt-4-turbo-preview",
  index = NULL,
  add_context = TRUE,
  chat_history = NULL,
  history_name = "chat_history",
  session_history = NULL,
  add_history = TRUE,
  task = "Context Only",
  k_context = 4,
  k_history = 4,
  save_history = TRUE,
  overwrite = FALSE,
  local = FALSE,
  embedding_model = NULL,
  stream = FALSE,
  rv = NULL
)
```

Arguments

`query`	The input query to be processed.
`service`	Name of the AI service to use, defaults to openai.
`model`	Name of the openai model to use, defaults to gpt-3.5-turbo
`index`	Index to look for context.
`add_context`	Whether to add context to the query. Options are `"always"`, `"sometimes"`, and `"never"`. The default is `"sometimes"`.
`chat_history`	Chat history dataframe for reference.
`history_name`	Name of the file where chat history is stored.
`session_history`	Session history data for reference.
`add_history`	Whether to add chat history to the query or not. Default is TRUE.

`task`	Task type, either "Context Only" or "Permissive Chat". Default is "Context Only".
`k_context`	Number of top context matches to consider. Default is 4.
`k_history`	Number of top chat history matches to consider. Default is 4.
`save_history`	Whether to save the chat history or not. Default is TRUE.
`overwrite`	Whether to overwrite the history file or not. Default is FALSE.
`local`	Whether to use the local model or not. Default is FALSE.
`embedding_model`	A model object to use for embedding. Only needed if local is TRUE. Default is NULL.
`stream`	Whether to stream the response or not. Default is FALSE.
`rv`	A reactive value to store the response. Default is NULL.

Value

A list containing the prompt, context, and answer.

Examples

```
rlang::is_interactive()
query <- "What is the capital of France?"
result <- chat_with_context(query = query, context = context)
```

`chat_with_retrieval` *Run Chat with Retrieval*

Description

Run the Chat with Retrieval shiny app

Usage

```
chat_with_retrieval()
```

Value

This function has no return value.

Examples

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

## End(Not run)
```

`collect_dataframes` *Collect Dataframes*

Description

Collect all the dataframes in the global environment.

Usage

```
collect_dataframes()
```

Value

A character vector of dataframe names.

`copilot_addin` *Copilot-style Code Suggestions*

Description

Copilot-style Code Suggestions

Usage

```
copilot_addin()
```

`crawl` *Scrape and process all hyperlinks within a given URL*

Description

This function scrapes all hyperlinks within a given URL and processes the data into a tibble format. It saves the resulting tibble into a parquet file.

Usage

```
crawl(
  url,
  index_create = TRUE,
  aggressive = FALSE,
  overwrite = FALSE,
  update = FALSE,
  pkg_version = "not a package",
  pkg_name = NULL,
  service = "openai"
)
```

Arguments

`url`	A character string with the URL to be scraped.
`index_create`	A logical value indicating whether to create an index. Default is TRUE.
`aggressive`	A logical value indicating whether to use aggressive link crawling. Default is FALSE.
`overwrite`	A logical value indicating whether to overwrite scraped pages and index if they already exist. Default is FALSE.
`update`	A logical value indicating whether to update scraped pages, defaults to FALSE.
`pkg_version`	Package version number
`pkg_name`	Package name
`service`	The service to use for scraping. Default is "openai". Options are "openai" and "local".

Value

NULL. The resulting tibble is saved into a parquet file.

`create_index_from_audio`

Create index from audio

Description

This function creates an index by first transcribing the audio file and then creating the index.

Usage

```
create_index_from_audio(file_path, source, link = NULL, overwrite = FALSE)
```

Arguments

`file_path`	Character string specifying the path to the audio file.
`source`	Character string specifying the source of the audio.
`link`	Character string specifying the link to the audio file (optional).
`overwrite`	Logical, whether to overwrite the existing index (default: FALSE).

Value

The function writes an index in Parquet format to disk.

`create_index_from_pdf` *Create index from PDF*

Description

This function creates an index by first ingesting the PDF file and then creating the index.

Usage

```
create_index_from_pdf(file_path, source, link = NULL, overwrite = FALSE)
```

Arguments

`file_path`	Character string specifying the path to the PDF file.
`source`	Character string specifying the source of the PDF.
`link`	Character string specifying the link to the PDF file (optional).
`overwrite`	Logical, whether to overwrite the existing index (default: FALSE).

Value

The function writes an index in Parquet format to disk.

`create_transcript` *Get Transcription from Audio*

Description

This function transcribes audio files and returns the transcription text.

Usage

```
create_transcript(file_path, prompt = NULL, chunk_size = 120)
```

Arguments

`file_path`	Character string specifying the path to the audio file.
`prompt`	Character string specifying the prompt for transcription (optional).
`chunk_size`	Audio size in seconds.

Value

A character string containing the transcription text.

delete_history *Delete chat history files*

Description

This function interactively deletes chat history files stored in the user's directory. It lists all the .parquet files in the gpttools data directory and prompts the user for confirmation before deleting each file.

Usage

```
delete_history(local = FALSE)
```

Arguments

`local`	Whether to delete history made with local or OpenAI embeddings.

Examples

```
## Not run:
# Interactively delete chat history files
delete_history()

## End(Not run)
```

delete_index *Delete an Index File*

Description

Interactively deletes a specified index file from a user-defined directory. Presents the user with a list of available index files and prompts for confirmation before deletion.

Usage

```
delete_index()
```

document_data *Document Data*

Description

This function runs a Shiny application to view and document data.

Usage

```
document_data()
```

`extract_code_chunks` *Extract Code Chunks from Text*

Description

This function extracts code chunks that are enclosed in triple backticks from a given text input. It supports identifying code blocks that are indicated by triple backticks followed by an optional language identifier. The function returns the extracted code chunks.

Usage

```
extract_code_chunks(text)
```

Arguments

`text`	A character vector containing the text from which code chunks need to be extracted.

Value

A list containing the extracted code chunks.

Examples

~~~
sample_text <- "Here is some text.
```
This is a code chunk.
```"
extract_code_chunks(sample_text)
~~~

`get_selection` *Wrapper around selectionGet to help with testthat*

Description

Wrapper around selectionGet to help with testthat

Usage

```
get_selection()
```

Value

Text selection via `rstudioapi::selectionGet`

get_transformer_model *Get Transformer Model*

Description

This function is designed to download and load a pre-trained transformer model using the transformers Python library via the reticulate package. It checks for the availability of the required Python package and then downloads the specified transformer model.

Usage

```
get_transformer_model(
  model_name = getOption("gpttools.local_embed_model", "BAAI/bge-small-en-v1.5")
)
```

Arguments

`model_name`	The name of the transformer model to download. This should be in the format "username/modelname" as recognized by the transformers library. Default is "BAAI/bge-small-en-v1.5".

Value

An object of the downloaded transformer model.

Note

Users of this function need to ensure that the Python environment is set up with the 'transformers' package installed. The function uses the 'reticulate' R package to interface with Python and the user may need to configure it accordingly.

Examples

```
## Not run:
# To get the default transformer model:
get_transformer_model()

# To get a custom transformer model by specifying the model name:
get_transformer_model("bert-base-uncased")

## End(Not run)
```

`ghost_chat` *Ghost Chat*

Description

Ghost Chat

Usage

```
ghost_chat(
  service = getOption("gpttools.service", "openai"),
  stream = TRUE,
  where = "source"
)
```

Arguments

`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.
`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")`.
`where`	A character string indicating the location or environment where the chat is taking place. Options are `c("console", "source", and "shiny")`. The default is `""`, which means the chat is taking place in the R console.

`ghost_writer` *Writing Assistant*

Description

Writing Assistant

Usage

```
ghost_writer(
  service = getOption("gpttools.service", "openai"),
  stream = TRUE,
  where = "source"
)
```

Arguments

`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.
`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")`.
`where`	A character string indicating the location or environment where the chat is taking place. Options are `c("console", "source", and "shiny")`. The default is `""`, which means the chat is taking place in the R console.

`gpttools_index_all_scraped_data` *Index All Scraped Data*

Description

This function iterates through all the text files in a specified directory, updating or creating indexes for each domain contained in the file names. Allows customization of the indexing process through various parameters.

Usage

```
gpttools_index_all_scraped_data(
  overwrite = FALSE,
  local_embeddings = TRUE,
  dont_ask = TRUE
)
```

Arguments

`overwrite`	A logical value determining whether to overwrite existing indexes.
`local_embeddings`	A logical indicating whether to use local embeddings for indexing.
`dont_ask`	A logical value that, if TRUE, disables interactive confirmation prompts during the indexing process.

Details

The function first retrieves a list of all text files in the targeted directory. For each file, it extracts the domain name from the filename, prints an informative message about the indexing process for that domain, and then proceeds to create or update the index for the domain based on the function arguments.

Value

Invisible NULL. The function is called for its side effects.

Examples

```
# Index all scraped data without overwriting existing indexes, using local
# embeddings, and without interactive prompts.

## Not run:
gpttools_index_all_scraped_data(
  overwrite = FALSE,
  local_embeddings = TRUE,
  dont_ask = TRUE
)

## End(Not run)
```

`gpt_sitrep` *GPT Sitrep*

Description

Check the status of the OpenAI API and RStudio API, and inform the user of the settings for gpttools.

Usage

```
gpt_sitrep()
```

Value

A message is returned to the user informing them of the status of the OpenAI API, RStudio API, and gpttools settings.

Examples

```
## Not run:
gpt_sitrep()

## End(Not run)
```

`ingest_pdf` *Ingest PDF and write to index*

Description

This function reads the text from a PDF file, processes it, and writes the result to an index.

Usage

```
ingest_pdf(file_path, source, link = NULL)
```

Arguments

`file_path`	Character string specifying the path to the PDF file.
`source`	Character string specifying the source of the PDF.
`link`	Character string specifying the link to the PDF file (optional).

Value

The function writes an index in Parquet format to disk.

`insert_text` *Wrapper around selectionGet to help with testthat*

Description

Wrapper around selectionGet to help with testthat

Usage

```
insert_text(improved_text)
```

Arguments

`improved_text`	Text from model queries to inert into script or document

install_sentence_transformers

Install sentence-transformers and its dependencies

Description

`install_sentence_transformers()` installs the sentence-transformers python package and its direct dependencies. This code is taken and only slightly modified from the tensorflow and reticulate packages. See the original code from tensorflow::install_tensorflow().

Usage

```
install_sentence_transformers(
  method = c("auto", "virtualenv", "conda"),
  conda = "auto",
  version = NULL,
  envname = NULL,
  extra_packages = NULL,
  restart_session = TRUE,
  conda_python_version = NULL,
  ...,
  pip_ignore_installed = TRUE,
  python_version = conda_python_version
)
```

Arguments

`method` Installation method. By default, "auto" automatically finds a method that will work in the local environment. Change the default to force a specific installation method. Note that the "virtualenv" method is not available on Windows.

`conda` The path to a conda executable. Use `"auto"` to allow `reticulate` to automatically find an appropriate conda binary. See **Finding Conda** and `conda_binary()` for more details.

`version` version to install. Valid values include:

- `"release"` installs the latest release version of sentence-transformers (which may be incompatible with the current version of the R package)
- A version specification like `"2.4"` or `"2.4.0"`. Note that if the patch version is not supplied, the latest patch release is installed (e.g., `"2.4"` today installs version "2.4.2")
- `nightly` for the latest available nightly build.
- The full URL or path to a installer binary or python *.whl file.

`envname` The name, or full path, of the environment in which Python packages are to be installed. When `NULL` (the default), the active environment as set by the `RETICULATE_PYTHON_ENV` variable will be used; if that is unset, then the `r-reticulate` environment will be used.

`extra_packages` Additional Python packages to install along with TensorFlow.

`restart_session`
: Restart R session after installing (note this will only occur within RStudio).

`...` other arguments passed to `reticulate::conda_install()` or `reticulate::virtualenv_install()`, depending on the `method` used.

`pip_ignore_installed`
: Whether pip should ignore installed python packages and reinstall all already installed python packages. This defaults to `TRUE`, to ensure that TensorFlow dependencies like NumPy are compatible with the prebuilt TensorFlow binaries.

`python_version`, `conda_python_version`
: Pass a string like "3.8" to request that conda install a specific Python version. This is ignored when attempting to install in a Python virtual environment.

Details

You may be prompted to download and install miniconda if reticulate did not find a non-system installation of python. Miniconda is the recommended installation method for most users, as it ensures that the R python installation is isolated from other python installations. All python packages will by default be installed into a self-contained conda or venv environment named "r-reticulate". Note that "conda" is the only supported method on M1 Mac. If you initially declined the miniconda installation prompt, you can later manually install miniconda by running `reticulate::install_miniconda()`.

Custom Installation

`install_sentence_transformers()` or isn't required to use tensorflow with the package. If you manually configure a python environment with the required dependencies, you can tell R to use it by pointing reticulate at it, commonly by setting an environment variable:

```
Sys.setenv("RETICULATE_PYTHON" = "~/path/to/python-env/bin/python")
```

Additional Packages

If you wish to add additional PyPI packages to your Keras / TensorFlow environment you can either specify the packages in the `extra_packages` argument of `install_sentence_transformers()`, or alternatively install them into an existing environment using the `reticulate::py_install()` function.

`launch_settings` *Settings App for gpttools*

Description

Settings App for gpttools

Usage

```
launch_settings()
```

Value

This function has no return value.

`list_index` *List Index Files*

Description

This function lists the index files in the specified directory.

Usage

```
list_index(dir = "index", full_path = FALSE)
```

Arguments

`dir`	Name of the directory, defaults to "index"
`full_path`	If TRUE, returns the full path to the index files.

Value

A character vector containing the names of the index files found in the specified directory.

Examples

```
## Not run:
list_index()

## End(Not run)
```

`load_index` *Load Index Data for a Domain*

Description

This function loads the index data for a given domain from a parquet file.

Usage

```
load_index(domain, local_embeddings = FALSE)
```

Arguments

`domain`	A character string indicating the name of the domain.
`local_embeddings`	A logical indicating whether to load the local embeddings or the OpenAI embeddings. Defaults to FALSE.

Value

A data frame containing the index data for the specified domain.

Examples

```
## Not run:
load_index("example_domain")

## End(Not run)
```

`prep_data_prompt` *Preps OpenAI model prompt for data documentation*

Description

Prepares data prompt by summarizing data and printing it

Usage

```
prep_data_prompt(data, method, prompt)
```

Arguments

`data`	A data.frame
`method`	A summarization method, one of "skimr", "glimpse", or "summary"
`prompt`	A prompt string

Value

A string

Examples

```
prep_data_prompt(
  data = mtcars,
  method = "skimr",
  prompt = "This is a test prompt."
)
```

`read_history` *Read chat history from a file*

Description

Read chat history from a file

Usage

```
read_history(file_name = "chat_history", local)
```

Arguments

`file_name`	Name of the chat history file to read from. Default is "chat_history".
`local`	Whether to read from history made with local or OpenAI embeddings.

Value

A dataframe containing the chat history or NULL if the file doesn't exist.

Examples

```
## Not run:
# Read chat history from the default file
history <- read_history()

## End(Not run)
```

`remove_lines_and_spaces` *Remove new lines from a character vector.*

Description

Removes all new line characters from a character vector.

Usage

```
remove_lines_and_spaces(serie)
```

Arguments

`serie`	A character vector.

Value

A character vector with all new line characters removed.

`repair_index_names` *Repair Index Names*

Description

This function scans index files in a specified directory, checks if they lack a "name" column, and attempts to repair them by adding a "name" column based on the file name or user input.

Usage

```
repair_index_names(local = TRUE)
```

Arguments

`local`	logical; if TRUE, the function operates on the local index directory, otherwise it uses the main index directory.

Value

The function does not return a value, but it modifies the index files in place by either adding a "name" column based on automated generation or user input.

Examples

```
## Not run:
repair_index_names(TRUE)
repair_index_names(FALSE)

## End(Not run)
```

`run_document_data` *Shiny app that supports Document Data addin*

Description

Shiny app that supports Document Data addin

Usage

```
run_document_data()
```

Value

Nothing is returned, a shiny app is run

`run_extracted_code` *Execute and Present Extracted Code Chunks*

Description

This function takes extracted code chunks and utilizes the `reprex` packages to run them. It then formats the output into a collapsible HTML section for easy sharing and viewing. Images within the output are converted to clickable links to enhance accessibility and compactness of the presentation.

Usage

```
run_extracted_code(code)
```

Arguments

`code`	A character vector of code to be executed and presented. The code is run with `reprex::reprex()`, capturing the outcome for presentation.

Value

An HTML-formatted string that includes the collapsible section for the code's output. This HTML string can be embedded in web pages, markdown documents, or viewed directly in browsers. The function is designed to work seamlessly in interactive environments, hence it might return the output invisibly.

Examples

```
code <- "plot(1:10)"
html_output <- run_extracted_code(code)
# The returned value is an HTML string with the plotted output presented within a
# collapsible section and any images replaced by links.
```

`run_select_pkgs_app` *Run a Shiny App to Select and Save Installed Packages*

Description

This function launches a Shiny application that allows users to select from a list of installed packages and save their selections.

Usage

```
run_select_pkgs_app()
```

Details

The application provides a sidebar for package selection and an action button to save the selected packages. It displays the selected packages in a data table.

Value

None The function is used for its side effect of launching a Shiny app and doesn't return anything.

Examples

```
run_select_pkgs_app()
```

---

save_user_config *Save user configuration settings for gpttools*

Description

This function saves the user's configuration settings for gpttools service to a YAML file.

Usage

```
save_user_config(
  service = "openai",
  model = "gpt-4-turbo-preview",
  task = "Permissive Chat",
  local_embed = FALSE,
  openai_embed_model = "text-embedding-3-small",
  local_embed_model = "BAAI/bge-small-en-v1.5",
  k_context = 4,
  k_history = 4,
  save_history = TRUE,
  add_context = "sometimes",
  sources = "All",
  run_code = FALSE,
  persist = TRUE
)
```

Arguments

`service`	The name of the service to use, default is "openai".
`model`	The model to use, default is "gpt-4-1106-preview".
`task`	The task to perform, default is "Permissive Chat".
`local_embed`	Whether to use local embedding model. Default is FALSE.
`openai_embed_model`	The OpenAI embeddings model to use, default is "text-embedding-3-small".

`local_embed_model`	The local embeddings model to use, default is "BAAI/bge-small-en-v1.5".
`k_context`	The amount of context to keep, default is 4.
`k_history`	The amount of history to keep, default is 4.
`save_history`	Logical indicating whether history should be saved, default is TRUE.
`add_context`	Whether to add context to the query. Options are `"always"`, `"sometimes"`, and `"never"`. The default is `"sometimes"`.
`sources`	The sources to use, default is "All".
`run_code`	Whether to execute generated code with `reprex::reprex()`, default is FALSE.
`persist`	Logical indicating whether to persist the settings, default is TRUE.

Value

Invisible NULL.

`scrape_pkg_sites` *Scrape packaging sites*

Description

Scrape packaging sites

Usage

```
scrape_pkg_sites(
  sites = get_pkgs_to_scrape(local = TRUE),
  service = "local",
  index_create = TRUE,
  overwrite = TRUE,
  parallel = FALSE
)
```

Arguments

`sites`	A data frame containing the package sites to be scraped. If not provided, it defaults to `get_pkgs_to_scrape(local = TRUE)`.
`service`	The service to be used for scraping, defaults to "local".
`index_create`	Logical indicating whether to create an index, defaults to `TRUE`.
`overwrite`	Logical indicating whether to overwrite existing content, defaults to `TRUE`.
`parallel`	Logical indicating whether to use parallel processing, defaults to `FALSE`.

Details

This function scrapes the websites for the packages specified in the `sites` dataframe. If `sites` is empty, it alerts the user with no packages to scrape and returns `NULL` invisibly. If the user confirms to proceed, it scrapes each package site using the supplied details.

Value

Invisible `NULL`. The function is called for its side effects.

Examples

```
scrape_pkg_sites()
```

`scrape_url` *Scrape text from a URL*

Description

This function scrapes the text from a URL and returns a character vector.

Usage

```
scrape_url(url)
```

Arguments

`url`	A character string containing a valid URL.

Value

A character vector containing the text from the URL.

`script_to_function_addin` *Convert Script to Function Addin*

Description

"convert this R code into an R function"

Usage

```
script_to_function_addin()
```

`set_user_config` *Set user configuration settings for gpttools from a file*

Description

This function sets the user's configuration settings for gpttools by reading a YAML file. If no path is provided, it will look for the file in the default R user config directory.

Usage

```
set_user_config(path = NULL)
```

Arguments

`path`	The path to the config YAML file. If NULL, the default path is used.

Value

Invisible NULL.

`suggest_code_improvements` *Suggest code improvements using a code copilot*

Description

This function provides suggestions for improving the provided code using an expert code copilot. The output is formatted for inclusion in an R file.

Usage

```
suggest_code_improvements()
```

Value

A string containing suggestions for code improvements, including descriptions of the changes and any necessary annotations as code comments. The output is formatted for inclusion in an R file.

`suggest_unit_test_addin` *Suggest a unit test for a function*

Description

Prompt: "Suggest a unit test for this function using the testthat package"

Usage

```
suggest_unit_test_addin()
```

`summarize_data` *Summarize data*

Description

Summarize a data frame using one of three methods.

Usage

```
summarize_data(
  data,
  method = c("skimr", "skimr_lite", "column_types", "summary")
)
```

Arguments

`data`	A data frame
`method`	A character vector specifying the method to use for summarizing the data. Must be one of "skimr", "skimr_lite", "column_types", or "summary". Default is "skimr".

Value

Summarized data according to specified method

`transcribe_audio` *Transcribe audio and write to index*

Description

This function transcribes audio files, processes the text, and writes the result to an index.

Usage

```
transcribe_audio(
  file_path,
  source = NA,
  link = NA,
  prompt = NA,
  chunk_size = 120
)
```

Arguments

`file_path`	Character string specifying the path to the audio file.
`source`	Character string specifying the source of the audio.
`link`	Character string specifying the HTML link to the audio file (optional).
`prompt`	Character string specifying the prompt for transcription (optional).
`chunk_size`	Audio size in seconds

Value

The function writes an index in Parquet format to disk.

Index