Package ‘PacketLLM’

April 24, 2025

Title Interactive 'OpenAI' Model Integration in 'RStudio'

Version 0.1.0

Description Offers an interactive 'RStudio' gadget interface for communicating with 'OpenAI' large language models (e.g., 'gpt-4o', 'gpt-4o-mini', 'gpt-4.1', 'o1', 'o3-mini') (<`https://platform.openai.com/docs/api-reference`>). Enables users to conduct multiple chat conversations simultaneously in separate tabs. Supports uploading local files (R, PDF, DOCX) to provide context for the models. Allows per-conversation configuration of model parameters such as temperature and system messages (where supported by the model). API interactions via the 'httr' package are performed asynchronously using 'promises' and 'future' to avoid blocking the R console. Useful for tasks like code generation, text summarization, and document analysis directly within the 'RStudio' environment. Requires an 'OpenAI' API key set as an environment variable.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

URL `https://github.com/AntoniCzolgowski/PacketLLM`

BugReports `https://github.com/AntoniCzolgowski/PacketLLM/issues`

Imports future, httr, pdftools, promises, readtext, shiny, shinyjs, stats, tools, utils

Depends R (>= 4.1.0)

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Antoni Czolgowski [aut, cre]

Maintainer Antoni Czolgowski <`antoni.czolgowski@gmail.com`>

Repository CRAN

Date/Publication 2025-04-24 17:10:06 UTC

Contents

`add_attachment_to_active_conversation`
Adds an attachment to the active conversation

Description

Associates a file's name and content with the currently active conversation. This function prevents adding attachments with duplicate names to the same conversation.

Usage

```
add_attachment_to_active_conversation(name, content)
```

Arguments

`name`	Character string. The name of the attachment file (e.g., "script.R").
`content`	Character string. The content of the file as a single string.

Value

Logical. `TRUE` if the attachment (consisting of `name` and `content`) was successfully added to the active conversation's attachment list. `FALSE` if no conversation is active, the active conversation doesn't exist anymore, or if an attachment with the same `name` already exists in the active conversation.

Examples

```
# Setup
reset_history_manager()
conv_addattach_id <- create_new_conversation(activate = TRUE)

# Add a first attachment
result1 <- add_attachment_to_active_conversation("report.txt", "Summary of findings.")
print(paste("Added first attachment:", result1)) # TRUE
print("Attachments after first add:")
print(get_active_conversation_attachments())

# Add a second, different attachment
result2 <- add_attachment_to_active_conversation("code.R", "x <- function(y) { y + 1 }")
print(paste("Added second attachment:", result2)) # TRUE
print("Attachments after second add:")
print(get_active_conversation_attachments())

# Try adding an attachment with the same name (should fail)
result3 <- add_attachment_to_active_conversation("report.txt", "Updated summary.")
print(paste("Added duplicate name attachment:", result3)) # FALSE
print("Attachments after duplicate attempt:")
print(get_active_conversation_attachments()) # Should be unchanged

# Try adding when no conversation is active
set_active_conversation(NULL)
result4 <- add_attachment_to_active_conversation("another.txt", "Content")
print(paste("Added attachment when none active:", result4)) # FALSE

# Clean up
reset_history_manager()
```

`add_message_to_active_history`

Adds a message to the active conversation's history

Description

Appends a message with the specified role and content to the history list of the currently active conversation. Handles automatic title generation on the first user message and locks the conversation model upon adding the first assistant message.

Usage

```
add_message_to_active_history(role, content)
```

Arguments

`role`	Character string. The role of the message author, must be one of "user", "assistant", or "system".
`content`	Character string. The content of the message.

Value

A list indicating the result of the operation. Possible structures: - `list(type = "title_set", new_title = "...")`: If this was the first user message and the title was automatically set. - `list(type = "assistant_locked_model")`: If this was the first assistant message, causing the model to be locked. - `list(type = "message_added")`: If a message was added without triggering title setting or model locking. - `list(type = "error", message = "...")`: If an error occurred (e.g., no active conversation, invalid role, conversation vanished).

Examples

```
# Setup
reset_history_manager()
conv_add_id <- create_new_conversation(activate = TRUE, title = "Initial Title")

# Add first user message (should set title)
result1 <- add_message_to_active_history(role = "user", content = "This is the very first post.")
print("Result after first user message:")
print(result1)
print(paste("New Title:", get_conversation_title(conv_add_id)))

# Add another user message (should just add message)
result2 <- add_message_to_active_history(role = "user", content = "Another question.")
print("Result after second user message:")
print(result2)

# Add first assistant message (should lock model)
result3 <- add_message_to_active_history(role = "assistant", content = "Here is the answer.")
print("Result after first assistant message:")
print(result3)
print(paste("Is model locked?", is_conversation_started(conv_add_id)))

# Add system message (just adds message)
result4 <- add_message_to_active_history(role = "system", content = "System notification.")
print("Result after system message:")
print(result4)
```

```
# Check final history
print("Final history:")
print(get_conversation_history(conv_add_id))

# Clean up
reset_history_manager()
```

add_user_message *Add user message to the active conversation Calls add_message_to_active_history, which handles model locking and title setting.*

Description

Add user message to the active conversation Calls add_message_to_active_history, which handles model locking and title setting.

Usage

```
add_user_message(text)
```

Arguments

`text`	The user's message text (a single character string).

Value

Invisible `NULL` (`invisible(NULL)`). This function is called for its side effect of adding a message to the conversation history. Stops with an error if `text` is not a single character string.

Examples

```
# This example modifies the internal state managed by history_manager.
# Ensure history_manager is initialized if running standalone.

# Setup: Create and activate a conversation
conv_id_user <- tryCatch(create_new_conversation(activate = TRUE), error = function(e) NULL)

if (!is.null(conv_id_user)) {
  # Add a message from the user
  add_user_message("Hello, this is my first message.")

  # Verify the message was added (optional)
  history <- get_active_chat_history()
  print(tail(history, 1)) # Show the last message

  # Clean up the conversation
  delete_conversation(conv_id_user)
} else {
  print("Skipping example as conversation setup failed.")
}

# Reset active conversation if needed
set_active_conversation(NULL)
```

`available_openai_models` *List of available OpenAI models for selection in the UI*

Description

List of available OpenAI models for selection in the UI

Usage

```
available_openai_models
```

Format

An object of class `character` of length 5.

`call_openai_chat` *Call OpenAI API*

Description

Function sends the conversation history to the OpenAI API and returns the model's response. For models in `simplified_models_list`, it does NOT send the `temperature` parameter.

Usage

```
call_openai_chat(messages, model, temperature = 0.5)
```

Arguments

`messages`	List of messages (each message is a list with 'role' and 'content' fields).
`model`	OpenAI model to use.
`temperature`	Temperature parameter (used only for models that support it).

Value

Character string containing the model's response text, or `NULL` if the response structure is unexpected. If an API or HTTP error occurs, the function stops execution with an error message.

Examples

```
## Not run:
  # This example requires the OPENAI_API_KEY environment variable to be set
  # and requires internet access to reach the OpenAI API.
  # Before running, ensure the key is set, e.g., using:
  # Sys.setenv(OPENAI_API_KEY = "your_actual_openai_api_key")
  # Remember to replace "your_actual_openai_api_key" with your real key.

  # 1. Define the conversation history
  example_messages <- list(
   list(role = "system", content = "You are a helpful assistant providing concise answers."),
    list(role = "user", content = "What is the main purpose of the 'httr' package in R?")
  )

  # 2. Choose a model (ensure it's in available_openai_models)
  selected_model <- "gpt-4o-mini" # Supports temperature

  # 3. Call the API using tryCatch for safety
  api_response <- tryCatch({
    call_openai_chat(
      messages = example_messages,
      model = selected_model,
      temperature = 0.7
     )
  }, error = function(e) {
    # Handle potential errors (API key missing, network issues, API errors)
    paste("API call failed:", e$message)
  })

  # 4. Print the response (or error message)
  print(api_response)

  # Example with a simplified model (omits temperature)
  selected_model_simple <- "o3-mini" # Does not support temperature

  # Check if this model is actually available in your package installation
  if(selected_model_simple %in% PacketLLM::available_openai_models) {
    api_response_simple <- tryCatch({
      call_openai_chat(
        messages = example_messages,
        model = selected_model_simple
        # Temperature argument is ignored internally by the function
      )
    }, error = function(e) {
      paste("API call failed:", e$message)
    })
    print(api_response_simple)
  } else {
     # Use message() for console output that can be suppressed if needed
     message(paste("Skipping simplified model example:",
                   selected_model_simple, "not in available_openai_models."))
  }
```

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

check_api_key *Check API Key*

Description

This function checks if the API key assigned to the `OPENAI_API_KEY` variable exists in the environment variables (e.g., in the .Renviron file). If the key is not set, the function will raise an error and stop execution. If the key is set, it returns TRUE invisibly.

Usage

```
check_api_key()
```

Value

Invisible `TRUE` (`invisible(TRUE)`) if the API key is set. Otherwise, stops execution with an error.

Examples

```
## Not run:
  # This function requires the OPENAI_API_KEY environment variable to be set.
  # You can check if the key is set using Sys.getenv("OPENAI_API_KEY").
  # If the key is set, calling check_api_key() will return TRUE invisibly.
  # If the key is NOT set, it will stop execution with an error message.

  # Example demonstrating how to handle the potential error if the key is missing:
  result <- tryCatch({
    check_api_key()
    "API key found." # Message if check passes
  }, error = function(e) {
    # Handle the error if the key is missing
    paste("Error:", e$message)
  })
  print(result)

## End(Not run)
```

`create_new_conversation`
Creates a new conversation

Description

Adds a new, empty conversation structure to the internal history store. Optionally sets the new conversation as the active one.

Usage

```
create_new_conversation(
  activate = FALSE,
  add_initial_settings = TRUE,
  title = NULL
)
```

Arguments

`activate`	Logical. Should the new conversation be set as active immediately? (Default: `FALSE`).
`add_initial_settings`	Logical. Should default settings (model, temperature, system message) be added to the conversation structure? (Default: `TRUE`).
`title`	Character string or `NULL`. An initial title for the conversation. If `NULL` (default), a title is generated based on the creation time.

Value

Character string. The unique ID assigned to the newly created conversation.

Examples

```
# Ensure manager is initialized (or reset)
reset_history_manager()
initialize_history_manager() # Creates one initial conversation
initial_active_id <- get_active_conversation_id()

# Create a new conversation without activating it
conv1_id <- create_new_conversation(activate = FALSE, title = "My First Topic")
print(paste("Created conv1 ID:", conv1_id))
current_active_id <- get_active_conversation_id() # Should still be the initial one
print(paste("Active ID:", current_active_id))

# Create another conversation and activate it immediately
conv2_id <- create_new_conversation(activate = TRUE, title = "My Second Topic")
print(paste("Created conv2 ID:", conv2_id))
current_active_id_2 <- get_active_conversation_id() # Should be conv2_id now
```

```
print(paste("Active ID:", current_active_id_2))

# Check total conversations
total_convs <- length(get_all_conversation_ids())
print(paste("Total conversations:", total_convs))

# Clean up by resetting (which deletes all)
reset_history_manager()
```

`delete_conversation` *Deletes the conversation with the given ID*

Description

Removes the specified conversation from the internal history store. If the deleted conversation was the active one, the active conversation ID is reset to `NULL`.

Usage

```
delete_conversation(id)
```

Arguments

`id`	Character string. The ID of the conversation to delete.

Value

Logical. `TRUE` if the conversation was found and successfully deleted. `FALSE` if no conversation with the specified `id` existed.

Examples

```
# Setup
reset_history_manager()
conv_del_id1 <- create_new_conversation(activate = FALSE, title = "To Delete")
conv_del_id2 <- create_new_conversation(activate = TRUE, title = "To Keep Active")
all_ids_before_del <- get_all_conversation_ids()
print(paste("Initial conversations:", paste(all_ids_before_del, collapse=", ")))
print(paste("Initial active ID:", get_active_conversation_id()))

# Delete the non-active conversation
deleted1 <- delete_conversation(conv_del_id1)
print(paste("Deleted conv_del_id1:", deleted1))
all_ids_after_del1 <- get_all_conversation_ids()
print(paste("Conversations after delete 1:", paste(all_ids_after_del1, collapse=", ")))
print(paste("Active ID after delete 1:", get_active_conversation_id())) # Should be unchanged

# Delete the active conversation
deleted2 <- delete_conversation(conv_del_id2)
print(paste("Deleted conv_del_id2:", deleted2))
```

```
all_ids_after_del2 <- get_all_conversation_ids() # Should be empty now
# MODIFIED LINE (was too long)
print(paste("Conversations after delete 2:", paste(all_ids_after_del2, collapse=", ")))
print(paste("Active ID after delete 2:", get_active_conversation_id())) # Should be NULL

# Try deleting a non-existent conversation
deleted3 <- delete_conversation("conv_non_existent")
print(paste("Deleted non-existent:", deleted3)) # FALSE

# Clean up
reset_history_manager()
```

---

get_active_chat_history

Gets the chat history for the active conversation

Description

Retrieves the list of messages associated with the currently active conversation.

Usage

```
get_active_chat_history()
```

Value

List. A list containing the message history (each element is a list with `role` and `content`) for the currently active conversation. Returns an empty list (`list()`) if no conversation is active or if the active conversation has no history yet.

Examples

```
# Setup
reset_history_manager()
conv_hist_id <- create_new_conversation(activate = TRUE)

# Get history when empty
print("Initial history:")
print(get_active_chat_history()) # list()

# Add messages
add_message_to_active_history("user", "Question 1")
add_message_to_active_history("assistant", "Answer 1")

# Get history after adding messages
print("History after messages:")
print(get_active_chat_history())

# Deactivate and check (should be empty list)
set_active_conversation(NULL)
```

```
print("History when none active:")
print(get_active_chat_history()) # list()

# Clean up
reset_history_manager()
```

`get_active_conversation`

Gets the full object of the active conversation

Description

Retrieves the complete data structure (a list) associated with the currently active conversation.

Usage

```
get_active_conversation()
```

Value

List or `NULL`. A list containing all data for the active conversation (id, title, history, attachments, settings, etc.), or `NULL` if no conversation is currently active.

Examples

```
# Setup
reset_history_manager()
conv_get_obj_id <- create_new_conversation(activate = TRUE, title = "Test Object")
add_message_to_active_history("user", "Message for object test")

# Get the active conversation object
active_obj <- get_active_conversation()
if (!is.null(active_obj)) {
  print("Active conversation object:")
  print(str(active_obj)) # Use str() for concise structure view
} else {
  print("No active conversation found.")
}

# Deactivate and try again
set_active_conversation(NULL)
active_obj_null <- get_active_conversation()
print(paste("Active object when none active:", is.null(active_obj_null))) # TRUE

# Clean up
reset_history_manager()
```

get_active_conversation_attachments

Gets the list of attachments for the active conversation

Description

Retrieves the list of attachments (files provided as context) associated with the currently active conversation.

Usage

```
get_active_conversation_attachments()
```

Value

List. A list where each element is a list containing `name` (character) and `content` (character) for an attachment associated with the currently active conversation. Returns an empty list (`list()`) if no conversation is active or if the active conversation has no attachments.

Examples

```
# Setup
reset_history_manager()
conv_getactiveattach_id <- create_new_conversation(activate = TRUE)

# Get attachments when none added
print("Attachments initially:")
print(get_active_conversation_attachments()) # list()

# Add some attachments
add_attachment_to_active_conversation("data.csv", "col1,col2\n1,2")
add_attachment_to_active_conversation("notes.txt", "Reminder")

# Get attachments again
print("Attachments after adding:")
attachments_list <- get_active_conversation_attachments()
print(attachments_list)
print(paste("Number of attachments:", length(attachments_list))) # 2

# Deactivate and check (should be empty list)
set_active_conversation(NULL)
print("Attachments when none active:")
print(get_active_conversation_attachments()) # list()

# Clean up
reset_history_manager()
```

`get_active_conversation_id`
Gets the ID of the active conversation

Description

Retrieves the identifier of the conversation currently marked as active.

Usage

```
get_active_conversation_id()
```

Value

Character string or `NULL`. The ID of the currently active conversation, or `NULL` if no conversation is active or if the previously active conversation ID points to a conversation that no longer exists.

Examples

```
# Setup
reset_history_manager()
conv_get_id <- create_new_conversation(activate = FALSE)

# Check when no conversation is active
print(paste("Active ID initially:", get_active_conversation_id())) # NULL

# Activate the conversation
set_active_conversation(conv_get_id)

# Get the active ID
print(paste("Active ID after set:", get_active_conversation_id())) # conv_get_id

# Deactivate
set_active_conversation(NULL)
print(paste("Active ID after unset:", get_active_conversation_id())) # NULL

# Clean up
reset_history_manager()
```

`get_all_conversation_ids`
Gets a list of IDs of all existing conversations

Description

Retrieves the unique identifiers for all conversations currently stored in the manager.

Usage

```
get_all_conversation_ids()
```

Value

Character vector. A vector containing the unique IDs of all currently stored conversations. Returns an empty character vector (`character(0)`) if no conversations exist.

Examples

```
# Setup
reset_history_manager()
initial_ids <- get_all_conversation_ids() # Should be character(0)
# MODIFIED LINE (was too long)
print(paste("IDs initially:", paste(initial_ids, collapse=",")))

# Create conversations
conv_all_id1 <- create_new_conversation()
conv_all_id2 <- create_new_conversation()

# Get all IDs
all_ids <- get_all_conversation_ids()
print(paste("IDs after creation:", paste(all_ids, collapse=",")))
print(paste("Number of conversations:", length(all_ids))) # 2

# Delete one and check again
delete_conversation(conv_all_id1)
ids_after_del <- get_all_conversation_ids() # Only ID2
print(paste("IDs after deletion:", paste(ids_after_del, collapse=",")))

# Clean up
reset_history_manager()
```

`get_assistant_response` *Get assistant response for the active conversation*

Description

Sends the prepared history (including system message and attachments for standard models) to the OpenAI API and returns the assistant's response. Handles model-specific logic internally.

Usage

```
get_assistant_response()
```

Value

Character string. Contains the assistant's response text. If an error occurs during message preparation or the API call, a descriptive error message (starting with "Error:" or "API Error:") is returned instead. Returns "Critical Error: No active conversation..." if no conversation is active.

Examples

```
## Not run:
# This function requires an active conversation with history,
# the OPENAI_API_KEY environment variable to be set, and internet access.

# Setup: Create, activate, and add a user message
conv_id_resp <- tryCatch(create_new_conversation(activate = TRUE), error = function(e) NULL)
if (!is.null(conv_id_resp)) {
  add_user_message("What day is it today?") # Add a user message first

  # Ensure the API key is set in your environment before running:
  # Sys.setenv(OPENAI_API_KEY = "your_actual_openai_api_key")

  # Get the response from the assistant
  # For less console output during standard use, you can set the global option:
  # options(PacketLLM.verbose = FALSE)
  assistant_reply <- get_assistant_response()
  # options(PacketLLM.verbose = TRUE) # Optionally set back for debugging

  # Print the response
  print(assistant_reply)

  # Verify the assistant's response was added to history (optional)
  # print(get_active_chat_history())

  # Clean up
  delete_conversation(conv_id_resp)
  set_active_conversation(NULL)
} else {
 print("Skipping example as conversation setup failed.")
}

## End(Not run)
```

get_conversation_attachments

Gets the list of attachments for the conversation with the given ID

Description

Retrieves the list of attachments (files provided as context) associated with a specific conversation ID.

Usage

```
get_conversation_attachments(id)
```

Arguments

`id`	Character string. The ID of the conversation.

Value

List or `NULL`. A list where each element is itself a list containing `name` (character) and `content` (character) for an attachment associated with the conversation specified by `id`. Returns `NULL` if the conversation does not exist. Returns an empty list (`list()`) if the conversation exists but has no attachments.

Examples

```
# Setup
reset_history_manager()
conv_attach_id <- create_new_conversation(activate = TRUE)

# Get attachments for new conversation (empty list)
print("Initial attachments by ID:")
print(get_conversation_attachments(conv_attach_id)) # list()

# Add an attachment using the exported function
add_attachment_to_active_conversation("file1.txt", "File content here")

# Get attachments again
print("Attachments after adding:")
print(get_conversation_attachments(conv_attach_id))

# Get attachments for non-existent ID
print("Attachments for non-existent:")
print(get_conversation_attachments("bad_id")) # NULL

# Clean up
reset_history_manager()
```

`get_conversation_data` *Gets the full conversation data object for the given ID*

Description

Retrieves the complete data structure (a list) associated with a specific conversation ID, including its history, attachments, settings, etc.

Usage

```
get_conversation_data(id)
```

Arguments

`id`	Character string. The ID of the conversation.

Value

List or `NULL`. A list containing all stored data associated with the conversation specified by `id`. Returns `NULL` if the conversation does not exist.

Examples

```
# Setup
reset_history_manager()
conv_getdata_id <- create_new_conversation(title = "Data Test")
set_conversation_temperature(conv_getdata_id, 0.9)

# Get conversation data by ID
data_obj <- get_conversation_data(conv_getdata_id)
if (!is.null(data_obj)) {
  print("Conversation data object:")
  print(str(data_obj))
}

# Get data for non-existent ID
print("Data for non-existent:")
print(get_conversation_data("bad_id")) # NULL

# Clean up
reset_history_manager()
```

`get_conversation_history` *Gets the chat history for the conversation with the given ID*

Description

Retrieves the list of messages associated with a specific conversation ID.

Usage

```
get_conversation_history(id)
```

Arguments

`id`	Character string. The ID of the conversation.

Value

List or `NULL`. A list containing the message history (each element is a list with `role` and `content`) for the conversation specified by `id`. Returns `NULL` if the conversation does not exist. Returns an empty list (`list()`) if the conversation exists but has no history yet.

Examples

```
# Setup
reset_history_manager()
conv_gethist_id <- create_new_conversation(activate = TRUE)

# Get history for new conversation
print("Initial history by ID:")
print(get_conversation_history(conv_gethist_id)) # list()

# Add messages using the exported function
add_message_to_active_history("user", "Hi there")
add_message_to_active_history("assistant", "Hello")

# Get history again
print("History after messages:")
print(get_conversation_history(conv_gethist_id))

# Get history for non-existent ID
print("History for non-existent:")
print(get_conversation_history("bad_id")) # NULL

# Clean up
reset_history_manager()
```

`get_conversation_model` *Gets the model name for the conversation with the given ID*

Description

Retrieves the name of the language model assigned to a specific conversation.

Usage

```
get_conversation_model(id)
```

Arguments

`id`	Character string. The ID of the conversation.

Value

Character string or `NULL`. The name of the OpenAI model assigned to the conversation with the specified `id`. Returns `NULL` if the conversation does not exist. Returns a fallback model name if the model field happens to be `NULL` internally.

Examples

```
# Setup
reset_history_manager()
conv_model_id <- create_new_conversation() # Uses default model from initialization

# Get the model for the conversation
model_name <- get_conversation_model(conv_model_id)
print(paste("Model for conversation:", model_name))

# Check a non-existent conversation
print(paste("Model for non-existent:", get_conversation_model("bad_id"))) # NULL

# Clean up
reset_history_manager()
```

---

get_conversation_title

Gets the title of the conversation with the given ID

Description

Retrieves the title associated with a specific conversation ID.

Usage

```
get_conversation_title(id)
```

Arguments

`id`	Character string. The ID of the conversation whose title is requested.

Value

Character string or `NULL`. The title of the conversation associated with the specified `id`. Returns `NULL` if the conversation does not exist. Returns a placeholder string like '`[No Title - ID: ...]`' if the title field happens to be `NULL` internally (should not normally occur).

Examples

```
# Setup
reset_history_manager()
conv_title_id1 <- create_new_conversation(title = "Specific Title")
conv_title_id2 <- create_new_conversation() # Default title generated

# Get title by ID
print(paste("Title for ID1:", get_conversation_title(conv_title_id1)))
print(paste("Title for ID2:", get_conversation_title(conv_title_id2)))

# Get title for non-existent ID
```

```
print(paste("Title for non-existent:", get_conversation_title("bad_id"))) # NULL

# Clean up
reset_history_manager()
```

`initialize_history_manager` *Initializes the history manager*

Description

Clears the current history state (all conversations and settings) and creates a single new, empty conversation, setting it as the active conversation. Optionally prints a message to the console in interactive sessions.

Usage

```
initialize_history_manager()
```

Value

Character string. The ID of the first, automatically created conversation after initialization.

Examples

```
# Initialize the manager. A message might appear in the console if run interactively.
first_conv_id <- initialize_history_manager()
print(paste("First conversation ID:", first_conv_id))

# Verify initialization
active_id_after_init <- get_active_conversation_id() # Should be first_conv_id
print(paste("Active ID after init:", active_id_after_init))
all_ids_after_init <- get_all_conversation_ids() # Should have 1 element
print(paste("Total conversations after init:", length(all_ids_after_init)))

# Clean up (reset state for other examples if needed)
reset_history_manager()
```

`is_conversation_started` *Checks if the conversation has started (model locked)*

Description

Determines if the model for a given conversation is locked, which typically occurs after the first assistant message has been added. Once locked, the model for the conversation usually cannot be changed.

Usage

```
is_conversation_started(id)
```

Arguments

`id`	Character string. The ID of the conversation to check.

Value

Logical. `TRUE` if the model for the conversation is locked, `FALSE` otherwise or if the conversation with the specified `id` does not exist.

Examples

```
# Setup
reset_history_manager()
conv_lock_id <- create_new_conversation(activate = TRUE)

# Check status of a new conversation (should be FALSE)
print(paste("Locked initially:", is_conversation_started(conv_lock_id)))

# Add a user message (does NOT lock the model)
add_message_to_active_history(role = "user", content = "First message")
print(paste("Locked after user msg:", is_conversation_started(conv_lock_id)))

# Add an assistant message (using the internal function locks the model)
add_message_to_active_history(role = "assistant", content = "Assistant reply")

# Check status after assistant message (should be TRUE)
print(paste("Locked after assistant msg:", is_conversation_started(conv_lock_id)))

# Check non-existent conversation
print(paste("Locked for non-existent:", is_conversation_started("conv_non_existent"))) # FALSE

# Clean up
reset_history_manager()
```

`parse_pages` *Parse page range*

Description

This function processes a character string specifying a page range (e.g., "1-3,5") and returns a numeric vector containing the individual page numbers, sorted and unique.

Usage

```
parse_pages(pages_str)
```

Arguments

`pages_str`	Character string specifying pages, e.g., "1-3,5".

Value

A numeric vector containing the unique page numbers specified in the input string, sorted in ascending order. Returns an empty integer vector if the input string is empty or contains only whitespace. Stops with an error if the input `pages_str` is not a single character string or if the format within the string is invalid (e.g., non-numeric parts, invalid ranges).

Examples

```
# Example 1: Simple range and single page
page_string1 <- "1-3, 5"
parsed_pages1 <- parse_pages(page_string1)
print(parsed_pages1) # Output: [1] 1 2 3 5

# Example 2: Multiple ranges and single pages, with spaces and duplicates
page_string2 <- " 2, 4-6, 9 , 11-12, 5 "
parsed_pages2 <- parse_pages(page_string2)
print(parsed_pages2) # Output: [1] 2 4 5 6 9 11 12 (sorted, unique)

# Example 3: Single number
page_string3 <- "10"
parsed_pages3 <- parse_pages(page_string3)
print(parsed_pages3) # Output: [1] 10

# Example 4: Empty string input
page_string_empty <- ""
parsed_pages_empty <- parse_pages(page_string_empty)
print(parsed_pages_empty) # Output: integer(0)

# Example 5: Invalid input (non-numeric) - demonstrates error handling
page_string_invalid <- "1-3, five"
## Not run:
# This will stop with an error message about "five"
tryCatch(parse_pages(page_string_invalid), error = function(e) print(e$message))

## End(Not run)

# Example 6: Invalid range format (missing end) - demonstrates error handling
page_string_invalid_range <- "1-"
## Not run:
# This will stop with an error message about invalid range format
tryCatch(parse_pages(page_string_invalid_range), error = function(e) print(e$message))

## End(Not run)

# Example 7: Invalid range format (start > end) - demonstrates error handling
page_string_invalid_order <- "5-3"
## Not run:
# This will stop with an error message about invalid range values
```

```
tryCatch(parse_pages(page_string_invalid_order), error = function(e) print(e$message))

## End(Not run)
```

read_file_content *Read file content*

Description

This function reads the content of a file with the extension .R, .pdf, or .docx and returns it as a single character string. TXT files are also supported. For PDF files, if the `pages` parameter is provided, only the selected pages will be read.

Usage

```
read_file_content(file_path, pages = NULL)
```

Arguments

`file_path`	Character string. Path to the file.
`pages`	Optional. A numeric vector specifying which pages (for PDF) should be read.

Value

A character string containing the file content, with pages separated by double newlines for PDF files. Stops with an error if the file does not exist, the format is unsupported, or required packages (`pdftools` for PDF, `readtext` for DOCX) are not installed or if `pages` is not numeric when provided.

Examples

```
# --- Example for reading an R file ---
# Create a temporary R file
temp_r_file <- tempfile(fileext = ".R")
writeLines(c("x <- 1", "print(x + 1)"), temp_r_file)

# Read the content
r_content <- tryCatch(read_file_content(temp_r_file), error = function(e) e$message)
print(r_content)

# Clean up the temporary file
unlink(temp_r_file)

# --- Example for reading a TXT file ---
temp_txt_file <- tempfile(fileext = ".txt")
writeLines(c("Line one.", "Second line."), temp_txt_file)
txt_content <- tryCatch(read_file_content(temp_txt_file), error = function(e) e$message)
print(txt_content)
unlink(temp_txt_file)
```

```
# --- Example for PDF (requires pdftools, only run if installed) ---
## Not run:
# This part requires the 'pdftools' package and a valid PDF file.
# Provide a path to an actual PDF file to test this functionality.
# Replace "path/to/your/sample.pdf" with a real path.

pdf_file_path <- "path/to/your/sample.pdf"

# Check if pdftools is installed and the file exists
if (requireNamespace("pdftools", quietly = TRUE) && file.exists(pdf_file_path)) {

  # Example: Read all pages
  pdf_content_all <- tryCatch(
    read_file_content(pdf_file_path),
    error = function(e) paste("Error reading all pages:", e$message)
  )
  # print(substr(pdf_content_all, 1, 100)) # Print first 100 chars

  # Example: Read only page 1
  pdf_content_page1 <- tryCatch(
    read_file_content(pdf_file_path, pages = 1),
    error = function(e) paste("Error reading page 1:", e$message)
  )
  # print(pdf_content_page1)

} else if (!requireNamespace("pdftools", quietly = TRUE)) {
  message("Skipping PDF example: 'pdftools' package not installed.")
} else {
  message("Skipping PDF example: File not found at '", pdf_file_path, "'")
}

## End(Not run)
# Note: Reading DOCX files is also supported if the 'readtext' package
# is installed, but a simple runnable example is difficult to create
# without including a sample file or complex setup.
```

`reset_history_manager` *Resets the entire state of the history manager*

Description

Clears all stored conversations, resets the active conversation ID to `NULL`, and resets the internal conversation counter used for generating IDs. Effectively returns the manager to its initial empty state. Optionally prints a message to the console in interactive sessions.

Usage

```
reset_history_manager()
```

Value

Invisible `NULL` (`invisible(NULL)`). Called for its side effect of clearing the history state.

Examples

```
# Setup: Initialize and add some data
reset_history_manager() # Ensure clean start
initialize_history_manager()
conv_reset_id <- create_new_conversation(activate = TRUE)
add_message_to_active_history("user", "Message before reset")
# MODIFIED LINE (was too long)
conv_count_before <- length(get_all_conversation_ids()) # Should be 2 initially
print(paste("Conversations before reset:", conv_count_before))
print(paste("Active ID before reset:", get_active_conversation_id())) # ID of conv_reset_id

# Reset the manager. A message might appear in the console if run interactively.
reset_history_manager()

# Verify state after reset
print(paste("Conversations after reset:", length(get_all_conversation_ids()))) # 0
print(paste("Active ID after reset:", get_active_conversation_id())) # NULL

# Note: After reset, you might need to initialize again if needed for subsequent operations
# initialize_history_manager()
```

`run_llm_chat_app` *Run the LLM Chat Application in RStudio Window*

Description

Launches the Shiny application as a Gadget in the RStudio Viewer pane, from where it can be opened in a separate window ("Show in new window"). The application allows interaction with LLM models, managing conversations, attachments, and settings, without blocking the RStudio console when opened in a new window.

Usage

```
run_llm_chat_app()
```

Value

Returns the value passed to `shiny::stopApp()` when the application is closed (typically `NULL`). The primary purpose is the side effect of launching the interactive application.

Examples

```
## Not run:
# This function launches an interactive Shiny gadget.
# It should be run in an interactive R session, preferably within RStudio.

# Ensure necessary setup like the OpenAI API key environment variable
# might be needed for the application's full functionality once launched.
# Example: Sys.setenv(OPENAI_API_KEY = "your_actual_openai_api_key")

# Launch the application
run_llm_chat_app()

# The application will open in the RStudio Viewer or a separate window.
# Interaction happens within the app's UI.
# To stop the app, close its window or click the 'X' button in the app's UI.

## End(Not run)
```

`set_active_conversation` *Sets the active conversation*

Description

Designates a conversation, specified by its ID, as the currently active one. Setting `id` to `NULL` deactivates any currently active conversation.

Usage

```
set_active_conversation(id)
```

Arguments

`id`	Character string (the ID of the conversation to activate) or `NULL`.

Value

Invisible `NULL` (`invisible(NULL)`). This function is called for its side effect of changing the active conversation state. It produces a warning if attempting to activate a non-existent conversation ID.

Examples

```
# Setup
reset_history_manager()
conv_set_id1 <- create_new_conversation(activate = FALSE, title = "Conv 1")
conv_set_id2 <- create_new_conversation(activate = FALSE, title = "Conv 2")
print(paste("Initial active ID:", get_active_conversation_id())) # NULL

# Set conv1 as active
```

```
set_active_conversation(conv_set_id1)
print(paste("Active ID after set 1:", get_active_conversation_id())) # conv_set_id1

# Set conv2 as active
set_active_conversation(conv_set_id2)
print(paste("Active ID after set 2:", get_active_conversation_id())) # conv_set_id2

# Set non-existent ID (should warn and not change active ID)
set_active_conversation("conv_non_existent")
active_after_invalid_set <- get_active_conversation_id() # Still conv_set_id2
print(paste("Active ID after set non-existent:", active_after_invalid_set))

# Deactivate by setting to NULL
set_active_conversation(NULL)
print(paste("Active ID after set NULL:", get_active_conversation_id())) # NULL

# Clean up
reset_history_manager()
```

set_conversation_model

Sets the model for the conversation, if it hasn't started

Description

Assigns a specified OpenAI language model to a conversation, but only if the conversation exists and has not yet "started" (i.e., no assistant messages have been added, `is_conversation_started(id)` is `FALSE`). The model name must be one of the available models listed in `available_openai_models`.

Usage

```
set_conversation_model(id, model_name)
```

Arguments

`id`	Character string. The ID of the conversation.
`model_name`	Character string. The name of the new model (must be one of `PacketLLM::available_openai_models`).

Value

Logical. `TRUE` if the model was successfully set for the conversation. `FALSE` if the conversation does not exist, the conversation has already started (model is locked), or the `model_name` is not valid/available.

Examples

```
# Setup
reset_history_manager()
conv_set_model_id <- create_new_conversation(activate = TRUE)
print(paste("Initial model:", get_conversation_model(conv_set_model_id)))

# Set a new valid model (use a known available model)
# Ensure the model exists in PacketLLM::available_openai_models
target_model <- "gpt-4o-mini" # Assuming this is usually available
if (target_model %in% PacketLLM::available_openai_models) {
  result_set <- set_conversation_model(conv_set_model_id, target_model)
  print(paste("Model set successful:", result_set))
  print(paste("Model after set:", get_conversation_model(conv_set_model_id)))
} else {
  message(paste("Skipping set model example: Target model", target_model, "not in list."))
}

# Try setting an invalid model name
result_invalid <- set_conversation_model(conv_set_model_id, "invalid-model-name")
print(paste("Invalid model set successful:", result_invalid)) # FALSE
model_after_invalid <- get_conversation_model(conv_set_model_id) # Unchanged
print(paste("Model after invalid set:", model_after_invalid))

# Simulate conversation start by adding an assistant message
add_message_to_active_history("user", "Lock question")
add_message_to_active_history("assistant", "Lock answer") # This locks it

# Try setting model after lock (should fail)
result_locked <- set_conversation_model(conv_set_model_id, "gpt-4o") # Try setting back
print(paste("Model set after lock successful:", result_locked)) # FALSE
model_after_locked <- get_conversation_model(conv_set_model_id) # Unchanged
print(paste("Model after locked attempt:", model_after_locked))

# Clean up
reset_history_manager()
```

`set_conversation_system_message`
Sets the system message for the conversation with the given ID

Description

Updates the system message (instructions provided to the language model) for a specific conversation.

Usage

```
set_conversation_system_message(id, message)
```

Arguments

`id`	Character string. The ID of the conversation.
`message`	Character string. The new system message content (must be a single string).

Value

Logical. `TRUE` if the system message was successfully updated. `FALSE` if the conversation does not exist or if the provided `message` is not a single character string.

Examples

```
# Setup
reset_history_manager()
conv_sys_id <- create_new_conversation()
initial_sys_msg <- get_conversation_data(conv_sys_id)$system_message
print(paste("Initial system message:", initial_sys_msg)) # Default message

# Set a valid system message
new_message <- "You are an expert R programmer. Respond only with code."
result_valid <- set_conversation_system_message(conv_sys_id, new_message)
print(paste("Valid set successful:", result_valid)) # TRUE
msg_after_set <- get_conversation_data(conv_sys_id)$system_message
print(paste("System message after set:", msg_after_set))

# Try setting an invalid message (e.g., not a single string)
result_invalid <- set_conversation_system_message(conv_sys_id, list("not a string"))
print(paste("Invalid set successful:", result_invalid)) # FALSE

# Try setting an invalid message (vector of strings)
result_invalid_vec <- set_conversation_system_message(conv_sys_id, c("Line 1", "Line 2"))
print(paste("Invalid vector set successful:", result_invalid_vec)) # FALSE

# Check message after invalid attempts
final_msg_after_invalid <- get_conversation_data(conv_sys_id)$system_message # Unchanged
print(paste("System message after invalid attempts:", final_msg_after_invalid))

# Clean up
reset_history_manager()
```

`set_conversation_temperature` *Sets the temperature for the conversation with the given ID*

Description

Updates the temperature setting (controls creativity/randomness of responses) for a specific conversation.

Usage

```
set_conversation_temperature(id, temperature)
```

Arguments

`id`	Character string. The ID of the conversation.
`temperature`	Numeric. The new temperature value, must be a single number between 0 and 1 (inclusive).

Value

Logical. `TRUE` if the temperature was successfully updated. `FALSE` if the conversation does not exist or if the provided `temperature` value is invalid (not a single number between 0 and 1).

Examples

```
# Setup
reset_history_manager()
conv_temp_id <- create_new_conversation()
initial_temp <- get_conversation_data(conv_temp_id)$temperature
print(paste("Initial temperature:", initial_temp)) # Default temp

# Set a valid temperature
result_valid <- set_conversation_temperature(conv_temp_id, 0.85)
print(paste("Valid set successful:", result_valid)) # TRUE
temp_after_valid <- get_conversation_data(conv_temp_id)$temperature # 0.85
print(paste("Temp after valid set:", temp_after_valid))

# Set an invalid temperature (outside 0-1)
result_invalid <- set_conversation_temperature(conv_temp_id, 1.5)
print(paste("Invalid set successful:", result_invalid)) # FALSE
# MODIFIED LINE (was too long)
temp_after_invalid <- get_conversation_data(conv_temp_id)$temperature # Unchanged (0.85)
print(paste("Temp after invalid set:", temp_after_invalid))

# Set an invalid temperature (wrong type)
result_invalid_type <- set_conversation_temperature(conv_temp_id, "high")
print(paste("Invalid type set successful:", result_invalid_type)) # FALSE

# Try on non-existent ID
result_bad_id <- set_conversation_temperature("bad_id", 0.5)
print(paste("Set on bad ID successful:", result_bad_id)) # FALSE

# Clean up
reset_history_manager()
```

Index