Package ‘rsynthbio’

June 4, 2025

Type Package

Title Synthesize Bio API Wrapper

Version 2.0.0

Description Access Synthesize Bio models from their API <https://app.synthesize.bio/> using this wrapper that provides a convenient interface to the Synthesize Bio API, allowing users to generate realistic gene expression data based on specified biological conditions. This package enables researchers to easily access AI-generated transcriptomic data for various modalities including bulk RNA-seq, single-cell RNA-seq, microarray data, and more.

URL https://github.com/synthesizebio/rsynthbio

BugReports https://github.com/synthesizebio/rsynthbio/issues

Imports getPass, keyring, jsonlite, httr

Suggests rmarkdown, knitr, testthat (>= 3.0.0), mockery

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

VignetteBuilder knitr

License MIT + file LICENSE

NeedsCompilation no

Author Candace Savonen [aut, cre]

Maintainer Candace Savonen <cansav09@gmail.com>

Repository CRAN

Date/Publication 2025-06-04 08:00:06 UTC

Contents

`API_BASE_URL` *API Base URL*

Description

Base URL for the Synthesize Bio API

Usage

```
API_BASE_URL
```

Format

An object of class `character` of length 1.

`clear_synthesize_token` *Clear Synthesize Bio API Token*

Description

Clears the Synthesize Bio API token from the environment for the current R session. This is useful for security purposes when you've finished working with the API or when switching between different accounts.

Usage

```
clear_synthesize_token(remove_from_keyring = FALSE)
```

Arguments

`remove_from_keyring`	Logical, whether to also remove the token from the system keyring if it's stored there. Defaults to FALSE.

Value

Invisibly returns TRUE.

Examples

```
## Not run:
# Clear token from current session only
clear_synthesize_token()

# Clear token from both session and keyring
clear_synthesize_token(remove_from_keyring = TRUE)

## End(Not run)
```

`extract_expression_data` *Extract Gene Expression Data from API Response*

Description

Extracts and combines gene expression data from a complex API response, with proper formatting and metadata association.

Usage

```
extract_expression_data(parsed_content, as_counts = TRUE)
```

Arguments

`parsed_content`	The parsed API response list
`as_counts`	Logical, if FALSE, transforms the predicted expression counts into logCPM (default is TRUE, returning raw counts).

Value

A list with two components: - metadata: tibble containing sample metadata - expression: tibble containing combined gene expression data

get_valid_modalities *Get Valid Modalities*

Description

Returns a vector of possible output modalities for the supported model. These modalities represent different types of gene expression data that can be generated by the Synthesize Bio API. Note only version 2 can be accessed with this version of the package. If you would like to use v1 models return to 1.x.x versions of this package.

Usage

```
get_valid_modalities()
```

Value

A character vector containing the valid modality strings.

Examples

```
# Get all supported modalities
modalities <- get_valid_modalities()
print(modalities)

# Check if a specific modality is supported
"bulk_rna-seq" %in% get_valid_modalities()
```

get_valid_modes *Get Valid Modes*

Description

Returns a vector of possible modes for the supported model. These modes represent different types of gene expression data that can be generated by the Synthesize Bio API.

Usage

```
get_valid_modes()
```

Value

A character vector containing the valid mode strings.

Examples

```
# Get all supported modes
modes <- get_valid_modes()
print(modes)

# Check if a specific mode is supported
"sample generation" %in% get_valid_modes()
```

`get_valid_query` *Get Valid Query Example*

Description

Generates a sample query for prediction and validation for the v2.0 model. This function provides an example query structure that can be modified for specific needs. The sample query contains two example inputs: one for a cell line with CRISPR perturbation and another for a primary tissue sample with disease information.

Usage

```
get_valid_query()
```

Value

A list representing a valid query structure for v2.0.

Examples

```
# Get a sample query
query <- get_valid_query()

# Modify the query for a different modality
query$modality <- "bulk_rna-seq"

# Adjust the number of samples to generate
query$inputs[[1]]$num_samples <- 10
```

`has_synthesize_token` *Check if Synthesize Bio API Token is Set*

Description

Checks whether a Synthesize Bio API token is currently set in the environment. Useful for conditional code that requires an API token.

Usage

```
has_synthesize_token()
```

Value

Logical, TRUE if token is set, FALSE otherwise.

Examples

```
 ## Not run:
# Check if token is set
if (!has_synthesize_token()) {
  # Prompt for token if not set
  set_synthesize_token()
}

## End(Not run)
```

`load_synthesize_token_from_keyring` *Load Synthesize Bio API Token from Keyring*

Description

Loads the previously stored Synthesize Bio API token from the system keyring and sets it in the environment for the current session.

Usage

```
load_synthesize_token_from_keyring()
```

Value

Invisibly returns TRUE if successful, FALSE if token not found in keyring.

Examples

```
## Not run:
# Load token from keyring
load_synthesize_token_from_keyring()

## End(Not run)
```

`log_cpm` *Log CPM Transformation*

Description

Transforms raw counts expression data into log1p(CPM) (Counts Per Million). This is a common normalization method for gene expression data that accounts for library size differences and applies a log transformation to reduce the effect of outliers.

Usage

```
log_cpm(expression)
```

Arguments

`expression`	A data.frame containing raw counts expression data.

Value

A data.frame containing log1p(CPM) transformed data.

Examples

```
# Create a sample expression matrix with raw counts
raw_counts <- data.frame(
  gene1 = c(100, 200, 300),
  gene2 = c(50, 100, 150),
  gene3 = c(10, 20, 30)
)

# Transform to log CPM
log_cpm_data <- log_cpm(raw_counts)
print(log_cpm_data)
```

`MODEL_MODALITIES` *Model Modalities*

Description

A nested list containing supported modalities for different model versions + sra = this is bulk RNA-seq

Usage

```
MODEL_MODALITIES
```

Format

A nested list with structure: model type > version > modalities

`predict_query` *Predict Gene Expression*

Description

Sends a query to the Synthesize Bio API (v2.0) for prediction and retrieves gene expression samples. This function validates the query, sends it to the API, and processes the response into usable data frames.

Usage

```
predict_query(query, raw_response = FALSE, as_counts = TRUE)
```

Arguments

`query`	A list representing the query data to send to the API. Use 'get_valid_query()' to generate an example.
`raw_response`	If you do not want the gene expression data extracted from the JSON response set this to FALSE. Default is to return only the expression and metadata.
`as_counts`	passed to extract_expression() function. Logical, if FALSE, transforms the predicted expression counts into logCPM (default is TRUE, returning raw counts).

Value

A list with two data frames: - 'metadata': contains metadata for each sample - 'expression': contains expression data for each sample Throws an error If the API request fails or the response structure is invalid.

Examples

```
# Set your API key (in practice, use a more secure method)
## Not run:

# To start using pysynthbio, first you need to have an account with synthesize.bio.
# Go here to create one: https://app.synthesize.bio/

Sys.setenv(SYNTHESIZE_API_KEY = "your_api_key_here")

# Create a query
query <- get_valid_query()

# Request raw counts
result <- predict_query(query, as_counts = TRUE)

# Access the results
metadata <- result$metadata
expression <- result$expression
```

```
# Request log CPM transformed data
log_result <- predict_query(query, as_counts = FALSE)
log_expression <- log_result$expression

# Explore the top expressed genes in the first sample
head(sort(expression[1, ], decreasing = TRUE))

## End(Not run)
```

`set_synthesize_token` *Set Synthesize Bio API Token*

Description

Securely prompts for and stores the Synthesize Bio API token in the environment. This function uses getPass to securely handle the token input without displaying it in the console. The token is stored in the SYNTHESIZE_API_KEY environment variable for the current R session.

Usage

```
set_synthesize_token(use_keyring = FALSE, token = NULL)
```

Arguments

`use_keyring`	Logical, whether to also store the token securely in the system keyring for future sessions. Defaults to FALSE.
`token`	Character, optional. If provided, uses this token instead of prompting. This parameter should only be used in non-interactive scripts.

Value

Invisibly returns TRUE if successful.

Examples

```
# Interactive prompt for token
## Not run:
set_synthesize_token()

# Provide token directly (less secure, not recommended for interactive use)
set_synthesize_token(token = "your-token-here")

# Store in system keyring for future sessions
set_synthesize_token(use_keyring = TRUE)

## End(Not run)
```

`validate_modality` *Validate Query Modality*

Description

Validates that the modality specified in the query is allowed for the v2.0 model. This function checks that the ‘modality‘ value is one of the supported modalities.

Usage

```
validate_modality(query)
```

Arguments

`query`	A list containing the query data.

Value

Invisibly returns TRUE if validation passes. Throws an error If the modality key is missing or if the selected modality is not allowed.

Examples

```
# Create a valid query
query <- get_valid_query()
validate_modality(query) # Passes validation

# Example with invalid modality
## Not run:
invalid_query <- get_valid_query()
invalid_query$modality <- "unsupported_modality"
validate_modality(invalid_query) # Throws error for invalid modality

## End(Not run)
```

`validate_query` *Validate Query Structure*

Description

Validates the structure and contents of the query based on the v2.0 model. This function checks that the query is a list and contains all required keys.

Usage

```
validate_query(query)
```

Arguments

`query`	A list containing the query data.

Value

Invisibly returns TRUE if validation passes. Throws an error If the query structure is invalid or missing required keys.

Examples

```
# Create a valid query
query <- get_valid_query()
validate_query(query) # Passes validation

# Example with invalid query (missing required key)
## Not run:
invalid_query <- list(inputs = list(), mode = "mean estimation")
validate_query(invalid_query) # Throws error for missing modality

## End(Not run)
```

Index