Package 'rsynthbio'

October 29, 2025

Type Package

Title Synthesize Bio API Wrapper

Version 3.0.2

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 (>= 3.0.2), mockery

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

VignetteBuilder knitr

License MIT + file LICENSE

NeedsCompilation no

Author Candace Savonen [aut, cre]

Maintainer Candace Savonen <cansav09@gmail.com>

Repository CRAN

Date/Publication 2025-10-29 20:00:02 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)
```

DEFAULT_POLL_INTERVAL_SECONDS

Default Poll Interval

Description

Default polling interval (seconds) for async model queries

Usage

```
DEFAULT_POLL_INTERVAL_SECONDS
```

Format

An object of class `numeric` of length 1.

DEFAULT_POLL_TIMEOUT_SECONDS

Default Poll Timeout

Description

Default maximum timeout (seconds) for async model queries

Usage

```
DEFAULT_POLL_TIMEOUT_SECONDS
```

Format

An object of class numeric of length 1.

DEFAULT_TIMEOUT *Default Timeout*

Description

Default timeout (seconds) for outbound HTTP requests

Usage

```
DEFAULT_TIMEOUT
```

Format

An object of class numeric of length 1.

---

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: - metadata: data.frame containing sample metadata - expression: data.frame containing combined gene expression data - latents: data.frame containing embeddings (if requested)

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.

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" %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(modality)
```

Arguments

modality	Character string specifying the modality. Either "bulk" or "single-cell".

Value

A character vector containing the valid mode strings.

Examples

```
# Get all supported modes
modes <- get_valid_modes(modality = "bulk")
print(modes)

# Check if a specific mode is supported
"sample generation" %in% get_valid_modes(modality = "single-cell")
```

`get_valid_query` *Get Valid Query Example*

Description

Generates a sample query for prediction and validation for the 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.

The returned query includes:

- `modality`: The data type ("bulk" or "single-cell")
- `mode`: The prediction mode ("sample generation" for bulk, "mean estimation" for single-cell)
- `inputs`: A list of biological conditions with metadata and num_samples

Optional fields can be added to control model behavior:

- `total_count`: Library size
- `deterministic_latents`: If TRUE, produces deterministic outputs
- `seed`: Random seed for reproducibility

Usage

```
get_valid_query(modality = "bulk")
```

Arguments

`modality`	Character string specifying the modality. Either "bulk" or "single-cell". Default is "bulk".

Value

A list representing a valid query structure.

Examples

```
# Get a sample query for bulk RNA-seq
query <- get_valid_query()

# Get a sample query for single-cell RNA-seq
query_sc <- get_valid_query(modality = "single-cell")

# Modify the query structure
query$inputs[[1]]$num_samples <- 10

# Add optional parameters
query$deterministic_latents <- TRUE
query$total_count <- 5000000
```

---

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 + bulk = bulk RNA-seq + single-cell = single-cell RNA-seq

Usage

```
MODEL_MODALITIES
```

Format

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

---

MODEL_MODES *Model Modes*

Description

A nested list containing supported modes for different model versions + bulk = bulk RNA-seq + single-cell = single-cell RNA-seq

Usage

```
MODEL_MODES
```

Format

A nested list with structure: version > modality > mode

predict_query *Predict Gene Expression*

Description

Sends a query to the Synthesize Bio API 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,
  as_counts = TRUE,
  api_base_url = API_BASE_URL,
  poll_interval_seconds = DEFAULT_POLL_INTERVAL_SECONDS,
  poll_timeout_seconds = DEFAULT_POLL_TIMEOUT_SECONDS,
  return_download_url = FALSE
)
```

Arguments

query	A list representing the query data to send to the API. Use ‘get_valid_query()‘ to generate an example. The query supports additional optional fields: • ‘total_count‘ (integer): Library size used when converting predicted log CPM back to raw counts. Higher values scale counts up proportionally. • ‘deterministic_latents‘ (logical): If TRUE, the model uses the mean of each latent distribution instead of sampling, producing deterministic outputs for the same inputs. Useful for reproducibility. • ‘seed‘ (integer): Random seed for reproducibility.
as_counts	Logical, if FALSE, transforms the predicted expression counts into logCPM (default is TRUE, returning raw counts).
api_base_url	The base URL for the API server. Default is API_BASE_URL.
poll_interval_seconds	Seconds between polling attempts of the status endpoint. Default is DEFAULT_POLL_INTERVAL_SEC (2).
poll_timeout_seconds	Maximum total seconds to wait before timing out. Default is DEFAULT_POLL_TIMEOUT_SECONDS (900 = 15 minutes).
return_download_url	Logical, if TRUE, returns a list containing the signed download URL instead of parsing into data frames. Default is FALSE.

Value

A list. If ‘return_download_url‘ is ‘FALSE‘ (default), the list contains two data frames: ‘metadata‘ and ‘expression‘. If ‘TRUE‘, the list contains ‘download_url‘ and empty ‘metadata‘ and ‘expression‘ data frames.

Examples

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

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

set_synthesize_token()

# 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))

# Use deterministic latents for reproducible results
query$deterministic_latents <- TRUE
result_det <- predict_query(query)

# Specify a custom total count (library size)
query$total_count <- 5000000
result_custom <- predict_query(query)

## 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 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 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