Package ‘rapidcodeR’

March 10, 2026

Title Optimized Data Analysis System for AI-Based Text Processing

Version 0.1.0

Description Extracts machine-readable variables from natural language text using AI APIs. Optimized for speed and cost efficiency through parallel processing and direct CSV-formatted responses from language models. Supports multiple AI providers with robust error handling and automatic retry mechanisms for failed extractions.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Imports stringr, openai, groqR, dplyr, rlang, parallel, future, future.apply

Suggests testthat (>= 3.0.0), irr

Config/testthat/edition 3

NeedsCompilation no

Author Gabriel Lönn [aut, cre], Sebastian Schutte [ctb] (Original code and package idea provided by.)

Maintainer Gabriel Lönn <gablon@prio.org>

Repository CRAN

Date/Publication 2026-03-10 20:40:02 UTC

Contents

ask_groq 2
ask_openai 3
calculate_overlap 4
main_func 5
make_value_row 6
parallel_execute 8
set_api_specs 10
set_coding_instruction 11
set_parameters 12

Index 14

`ask_groq` *Send Prompt to GROQ AI API*

Description

This function sends a text prompt to the GROQ AI API and returns the model's response. It includes error handling and uses the groqR package for API communication.

Usage

```
ask_groq(prompt, temp, topp = 1, model, api_key)
```

Arguments

`prompt`	Character. The text prompt to send to the AI model.
`temp`	Numeric. Temperature parameter controlling randomness (0-1). Lower values make output more deterministic. Required (typically from set_api_specs()).
`topp`	Numeric. Top-p parameter controlling diversity (0-1). Default is 1.
`model`	Character. GROQ model to use. Required (typically from set_api_specs()).
`api_key`	Character. GROQ API key. Must be provided.

Details

The function uses the groqR package to communicate with GROQ's API. It includes error handling that returns NA if the API call fails. The function is optimized for text processing tasks with a maximum token limit of 5000.

Value

Character. The AI model's response to the prompt, or NA if an error occurs.

See Also

`ask_openai()`

Examples

```
## Not run:
# Set API key first
Sys.setenv(GROQ_API_KEY = "your_api_key")

# Send a simple prompt
response <- ask_groq("What is the capital of France?")

# Use different parameters
response <- ask_groq("Analyze this text", temp = 0.3, topp = 0.9)
```

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

---

ask_openai *Send Prompt to OpenAI API*

Description

This function sends a text prompt to the OpenAI API and returns the model's response. It includes robust error handling and uses the openai package for API communication.

Usage

```
ask_openai(prompt, temp = NULL, topp = 1, model = NULL)
```

Arguments

prompt	Character. The text prompt to send to the AI model.
temp	Numeric. Temperature parameter controlling randomness (0-1). Lower values make output more deterministic. If NULL, retrieved from set_api_specs().
topp	Numeric. Top-p parameter controlling diversity (0-1). Default is 1.
model	Character. OpenAI model to use. If NULL, retrieved from set_api_specs().

Details

The function uses the openai package to communicate with OpenAI's API. It includes comprehensive error handling that returns NA if the API call fails or returns an unexpected format. The function expects the OPENAI_API_KEY environment variable to be set via `set_api_specs()`.

Value

Character. The AI model's response to the prompt, or NA if an error occurs.

See Also

`ask_groq()`, `set_api_specs()`

Examples

```
## Not run:
# Set API specifications first
set_api_specs(provider = "OpenAI", model = "gpt-4", temp = 0.7, api_key = "your_api_key")

# Send a simple prompt
response <- ask_openai("What is the capital of France?")

# Use different parameters
```

```
response <- ask_openai("Analyze this text", temp = 0.3, topp = 0.9)

## End(Not run)
```

`calculate_overlap` *Calculate Inter-Rater Agreement Across Multiple Datasets*

Description

This function calculates the percentage overlap (agreement) between multiple datasets containing the same variables. It's designed to assess reliability and consistency when the same data is processed multiple times or by different systems/raters.

Usage

```
calculate_overlap(datasets, alpha = FALSE)
```

Arguments

`datasets`	List. A list of data frames to compare. Each data frame should have the same structure with variable columns (Var1, Var2, etc.).
`alpha`	Logical. If TRUE, also compute Krippendorff's Alpha (nominal) for each variable across the provided datasets. Defaults to FALSE.

Details

The function performs comprehensive overlap analysis:

- Finds common IDs across all datasets to ensure fair comparison
- Aligns datasets by ID and uses all columns for comparison
- Calculates pairwise agreement for all possible dataset combinations
- Computes variable-wise agreement percentages
- Returns average agreement across all dataset pairs

The function expects exactly the same data structure as produced by the main processing functions: N coded variables (Var1-VarN). Agreement is calculated as the percentage of cases where values match exactly.

Value

Matrix. If alpha = FALSE: one-row matrix with average percentage agreement for each variable (columns), rounded to 2 decimals. If alpha = TRUE: two-row matrix with row "Overlap" (percent agreement) and row "Alpha" (Krippendorff's Alpha for nominal data; 0-1), rounded to 3 decimals for Alpha.

See Also

parallel_execute()

Examples

```
## Not run:
# Compare three processing runs
run1 <- parallel_execute(my_data, slicing_n = 100, cores = 4)
run2 <- parallel_execute(my_data, slicing_n = 100, cores = 4)
# Switch to GROQ for comparison
set_api_specs(provider = "Groq", model = "llama-3.3-70b-versatile",
  temp = 0.3, api_key = "your_groq_key")
run3 <- parallel_execute(my_data, slicing_n = 100, cores = 4)

overlap_scores <- calculate_overlap(list(run1, run2, run3))
# Returns vector like: c(Var1 = 85.6, Var2 = 92.1, ...)

## End(Not run)
```

---

main_func — *Main Text Processing Function for AI-Based Variable Extraction*

Description

This function processes batches of text data using AI models to extract machine-readable variables. It implements robust error handling and retry logic to ensure reliable processing even with API failures.

Usage

```
main_func(df, to_code_max_id, n_post, provider, worker_env = NULL)
```

Arguments

df	Data frame. Input data subset containing text to be processed.
to_code_max_id	Integer. Maximum number of posts to process in this batch.
n_post	Integer. Number of posts to process per API call (typically 15).
provider	Character. AI provider to use, either "OpenAI" or "Groq".
worker_env	Environment or NULL. When running in a parallel worker, the worker's package environment. If NULL (default), .package_env is used.

Details

The function implements a robust processing loop that:

- Samples posts randomly to avoid processing order bias
- Makes API calls in manageable batches
- Validates and cleans AI responses
- Handles API failures with counter and early stopping
- Tracks missing IDs for potential reprocessing
- Collects dataframes into a list for later binding

The AI models are instructed to return data in CSV format for efficient parsing. The function expects responses with exactly N columns as set by set_parameters().

Value

List of data frames. Each element contains successfully processed results with extracted variables. Returns empty list if no successful processing.

See Also

gpt_func(), groq_func(), make_value_row()

Examples

```
## Not run:
# For set_parameters(n_variables = 6):
instruction <- "Extract variables: var1,var2,var3,var4,var5,var6"
result <- main_func(
  df = my_data_subset,
  to_code_max_id = 30,
  n_post = 15,
  provider = "OpenAI"
)
processed_data <- bind_rows(result)  # Combine all dataframes

## End(Not run)
```

---

make_value_row *Parse AI Response into Formatted Row String*

Description

This function takes a single AI response string and formats it into a standardized SQL-style row format for consistent data processing. It handles text sanitization and ensures the response has exactly N elements (N set by set_parameters()).

Usage

```
make_value_row(ai_response)
```

Arguments

`ai_response`	Character. A single response string from an AI model, typically containing semicolon-separated values.

Details

The function performs several formatting operations:

- Converts input to character format
- Replaces quotes with backticks to avoid SQL conflicts
- Splits on semicolons to extract individual values
- Trims whitespace from each value
- Pads with NA values if fewer than expected elements
- Validates that we have the expected number of elements
- Converts to lowercase and formats as SQL-style row

The function expects exactly N values as set by set_parameters().

Value

Character. A formatted row string in SQL format like "('val1','val2',...)", or NA if validation fails.

See Also

`main_func()`

Examples

```
## Not run:
# AI response with semicolon-separated values (assuming set_parameters(n_variables = 6))
ai_response <- "123;en;Hello world;positive;high;topic1"
formatted_row <- make_value_row(ai_response)
# Returns: "('123','en','hello world','positive','high','topic1')"

## End(Not run)
```

parallel_execute *Execute Parallel Text Processing with AI Models*

Description

This is the main function that orchestrates parallel processing of text data using AI APIs for variable extraction. It optimizes for speed and cost by distributing work across multiple cores and handling missing data efficiently.

Usage

```
parallel_execute(
  test_data,
  slicing_n,
  n_post = 15,
  cores = 8,
  seed = NULL,
  multi_core = FALSE,
  benchmarking = FALSE,
  verbose = TRUE
)
```

Arguments

`test_data`	Data frame. Input data containing text to be processed.
`slicing_n`	Integer. Number of rows to sample from the input data for processing. The batch size will be automatically calculated by dividing this by the number of cores.
`n_post`	Integer. Number of posts to process in each API call. Default is 15. Must be between 1 and 1000. Larger values may be more efficient but use more API credits.
`cores`	Integer. Number of CPU cores to use for parallel processing. Default is 8. The function will use the provider specified in set_api_specs() to determine whether to use OpenAI or GROQ services.
`seed`	Integer or NULL. Random seed for reproducible sampling. If NULL (default), sampling is random. If set to a number, ensures identical datasets across runs.
`multi_core`	Logical. If TRUE, uses multicore backend for parallel processing (faster but Unix/Mac only). If FALSE (default), uses multisession backend (works on all platforms including Windows).
`benchmarking`	Logical. If TRUE, returns processing time in seconds instead of the result data frame. Default is FALSE.
`verbose`	Logical. If TRUE (default), progress and status messages are printed via `message()`. If FALSE, such messages are suppressed.

Details

The function implements a multi-stage parallel processing workflow:

1. Validates parameters and tests API connectivity
2. Randomly samples data and splits into batches for parallel processing
3. Executes parallel processing using futures and tracks progress
4. Handles missing/failed extractions with a secondary processing stage
5. Combines and deduplicates results

The function requires a coding instruction to be set via `set_coding_instruction()` before execution. This instruction tells the AI how to format its responses.

Value

Data frame. Processed results containing extracted variables, with duplicates removed (when `multi_response=FALSE`) and sorted by ID.

See Also

`set_api_specs()`, `set_coding_instruction()`, `set_parameters()`

Examples

```
## Not run:
# Set up API specifications and coding instruction
set_api_specs(provider = "OpenAI", model = "gpt-4", temp = 0.7, api_key = "your_key")
set_coding_instruction("Extract sentiment: id,sentiment,confidence")

# Process data with random sampling (default)
results <- parallel_execute(
  test_data = my_data,
  slicing_n = 240,
  cores = 4
)

# Process data with reproducible sampling (for calculate_overlap)
results1 <- parallel_execute(
  test_data = my_data,
  slicing_n = 240,
  cores = 4,
  seed = 123
)

# Switch to GROQ for comparison
set_api_specs(provider = "Groq", model = "llama-3.3-70b-versatile",
  temp = 0.3, api_key = "your_groq_key")
results2 <- parallel_execute(
  test_data = my_data,
  slicing_n = 240,
  cores = 4,
```

```
  seed = 123
)

# Calculate overlap between identical datasets
overlap_scores <- calculate_overlap(list(results1, results2))


## End(Not run)
```

`set_api_specs` *Set API Specifications for AI Models*

Description

This function sets the API specifications including provider, model, temperature, and API key for AI model interactions. It replaces the need for separate set_api_keys calls and allows for more flexible configuration.

Usage

```
set_api_specs(provider, model, temp, api_key, multi_response = FALSE)
```

Arguments

`provider`	Character. The AI provider to use, either "OpenAI" or "Groq".
`model`	Character. The specific model to use (e.g., "gpt-4", "gpt-3.5-turbo", "llama-3.3-70b-versatile").
`temp`	Numeric. Temperature parameter controlling randomness (0-1). Lower values make output more deterministic.
`api_key`	Character. The API key for the specified provider.
`multi_response`	Logical. If TRUE, allows AI to return multiple rows per input (only works with n_post=1). If FALSE (default), AI must return exactly one row per input.

Details

This function stores the API specifications in the package's internal environment and sets the appropriate environment variables for API authentication. The specifications are automatically retrieved by processing functions.

Value

Character. A confirmation message.

See Also

`set_coding_instruction()`, `set_parameters()`

Examples

```
## Not run:
# Set OpenAI specifications
set_api_specs(
  provider = "OpenAI",
  model = "gpt-4",
  temp = 0.7,
  api_key = "your_openai_key"
)

# Set GROQ specifications
set_api_specs(
  provider = "Groq",
  model = "llama-3.3-70b-versatile",
  temp = 0.3,
  api_key = "your_groq_key"
)

## End(Not run)
```

---

set_coding_instruction

Set Coding Instructions for AI Models

Description

This function sets the coding instructions that will be used by AI models to extract machine-readable variables from natural language text. The instruction defines how the AI should format and structure its responses.

Usage

```
set_coding_instruction(instruction)
```

Arguments

`instruction`	Character. A string containing the coding instruction that tells the AI model how to process and format the text data. Should specify the expected output format (typically CSV-structured responses).

Details

The coding instruction is stored in the package's internal environment and is automatically retrieved by processing functions. This instruction typically defines:

- Expected output format (e.g., CSV structure)
- Variable definitions and coding schemes
- Response formatting requirements

Value

Character. A confirmation message indicating the instruction has been set.

Examples

```
## Not run:
instruction <- "Extract sentiment and topics from posts. Return as CSV: id,sentiment,topic"
set_coding_instruction(instruction)

## End(Not run)
```

---

set_parameters *Set Processing Parameters for AI Models*

Description

This function configures the parameters that will be used by AI models to extract machine-readable variables from natural language text.

Usage

```
set_parameters(n_variables = 9, id_column = 1, text_column = 2, sep = ";")
```

Arguments

n_variables	Integer. Number of columns to extract (default is 9). This determines the total number of columns in the final dataset. Must be between 1 and 20.
id_column	Integer. The column number containing the unique ID for each observation (default is 1). Must be a positive integer.
text_column	Integer. The column number containing the text content to be processed (default is 2). Must be a positive integer.
sep	Character. Separator used in API response parsing (default is ";"). Must be one of: ";" (semicolon), "," (comma), or "|" (vertical bar).

Details

The function sets the expected parameters in the package's internal environment. This affects:

- Response parsing and validation
- Column naming (Var1, Var2, ..., VarN)
- Data structure expectations across all processing functions
- Column identification for text content and unique IDs
- Separator used for parsing API responses

The package automatically adds a unique internal ID column to each row during processing. This internal ID is used for tracking and is removed from the final output.

The total expected columns will be n_variables:

- All columns: Var1, Var2, ..., VarN

Value

Character. A confirmation message indicating the parameters have been set.

Examples

```
## Not run:
# Extract 4 columns, ID in column 1, text in column 2, semicolon separator
set_parameters(n_variables = 4, id_column = 1, text_column = 2, sep = ";")

# Extract 10 columns, ID in column 3, text in column 5, comma separator
set_parameters(n_variables = 10, id_column = 3, text_column = 5, sep = ",")

# Extract 6 columns, ID in column 2, text in column 3, vertical bar separator
set_parameters(n_variables = 6, id_column = 2, text_column = 3, sep = "|")

## End(Not run)
```

Index

ask_groq, 2
ask_groq(), *3*
ask_openai, 3
ask_openai(), *2*

calculate_overlap, 4

gpt_func(), *6*
groq_func(), *6*

main_func, 5
main_func(), *7*
make_value_row, 6
make_value_row(), *6*

parallel_execute, 8
parallel_execute(), *5*

set_api_specs, 10
set_api_specs(), *3*, *9*
set_coding_instruction, 11
set_coding_instruction(), *9, 10*
set_parameters, 12
set_parameters(), *9, 10*