Package: kagiPro (via r-universe)

June 9, 2026

Title Kagi API Client for R

Version 0.4.1

Description User-friendly R client for the Kagi APIs (Search, Enrich, Summarizer, and FastGPT). Build endpoint-specific query objects, run single or batch requests with one reusable connection, and write reproducible JSON outputs for analysis pipelines. Includes optional graceful error handling with dummy outputs and JSON-to-parquet conversion for downstream workflows.

Author Rainer M. Krug [aut, cre]

Maintainer Rainer M. Krug <Rainer.Krug@Senckenberg.de>

License GPL (>= 3)

Encoding UTF-8

LazyData true

Depends R (>= 4.2.0)

Imports arrow, DBI, digest, dplyr, duckdb, future, future.apply, httr2, jsonlite, rlang, ragnar, utils

Suggests here, keyring, knitr, quarto, rmarkdown, testthat (>= 3.0.0), vcr

VignetteBuilder knitr, quarto

RoxygenNote 7.3.3

Config/testthat/edition 3

URL https://github.com/rkrug/kagiPro, https://rkrug.github.io/kagiPro/

BugReports https://github.com/rkrug/kagiPro/issues

Config/pak/sysreqs cmake libicu-dev libpng-dev libxml2-dev libssl-dev python3 xz-utils

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

Date/Publication 2026-04-10 14:28:52 UTC

RemoteUrl https://github.com/rkrug/kagiPro

RemoteRef main

RemoteSha e4efb6151af89c7a2f8ff4fbedf509fed279a7a5

Contents

`clean_request` *Clean JSON Request Data While Preserving Query Metadata*

Description

Remove JSON request data files from endpoint JSON folders in a project while preserving per-query metadata files (`_query_meta.json`).

Usage

```
clean_request(project_folder, dry_run = FALSE, verbose = TRUE)
```

Arguments

`project_folder`	Root project folder containing endpoint subfolders.
`dry_run`	Logical. If `TRUE`, do not delete anything and only report what would be removed.
`verbose`	Logical. If `TRUE`, print progress messages.

Details

This function is intended for reclaiming disk space while keeping enough metadata for `kagi_update_query()` reruns.

Value

A list with:

- `details`: data frame with per-query deletion counts/bytes
- `totals`: list with `files` and `bytes`
- `dry_run`: logical flag

Examples

```
## Not run:
clean_request("kagi_project", dry_run = TRUE)
clean_request("kagi_project", dry_run = FALSE)

## End(Not run)
```

`content_markdown` *Extract Downloaded Content to Markdown*

Description

Extract Downloaded Content to Markdown

Usage

```
content_markdown(
  project_folder,
  endpoint = NULL,
  query_name = NULL,
  text_root = "markdown",
  output_format = "markdown",
  workers = 4,
  verbose = FALSE,
  progress = interactive()
)
```

Arguments

`project_folder`	Root project folder containing endpoint subfolders.
`endpoint`	Optional endpoint selector (for example ‘"search"‘ or ‘"enrich_news"‘). If ‘NULL‘, all supported endpoints are considered.
`query_name`	Optional query selector. If ‘NULL‘, all query partitions are considered.
`text_root`	Root folder name used for extracted text outputs.
`output_format`	Output format. Only ‘"markdown"‘ is supported.
`workers`	Number of parallel workers to use for extraction.
`verbose`	Logical indicating whether progress messages should be shown.
`progress`	Logical indicating whether a progress bar should be shown.

Value

A data frame with extraction status and diagnostics columns: ‘endpoint‘, ‘id‘, ‘query‘, ‘text_path‘, ‘status‘, ‘error‘.

`download_content` *Download Endpoint Content for Abstract Generation*

Description

Download Endpoint Content for Abstract Generation

Usage

```
download_content(
  project_folder,
  endpoint = NULL,
  query_name = NULL,
  workers = 4,
  progress = interactive(),
  verbose = FALSE
)
```

Arguments

`project_folder`	Root project folder containing endpoint subfolders.
`endpoint`	Optional endpoint selector (for example ‘"search"‘ or ‘"enrich_news"‘). If ‘NULL‘, all supported endpoints are considered.
`query_name`	Optional query selector. If ‘NULL‘, all query partitions are considered.
`workers`	Number of parallel workers to use for downloads.
`progress`	Logical indicating whether a progress bar should be shown.
`verbose`	Logical indicating whether progress messages should be shown.

Value

A data frame with download status and paths.

kagi_connection *Construct a Kagi API connection*

Description

Build a typed S3 object of class `kagi_connection` which holds the basic configuration required to talk to the Kagi API. This includes the API base URL, authentication key, and retry settings.

Usage

```
kagi_connection(
  base_url = "https://kagi.com/api/v0",
  api_key = Sys.getenv("KAGI_API_KEY"),
  max_tries = 3
)
```

Arguments

`base_url`	Character scalar. Base URL for the Kagi API. Defaults to `"https://kagi.com/api/v0"`.
`api_key`	API key used for authentication. By default this is read from the environment variable `KAGI_API_KEY`. Best practice is to set this variable in your `~/.Renviron`. Advanced users may also supply a function that resolves the key lazily at request time (see `resolve_api_key()`).
`max_tries`	Integer scalar. Maximum number of retry attempts for transient errors. Defaults to 3.

Value

An object of class `kagi_connection` with components:

`base_url` Base API URL.

`api_key` API key (or a function to resolve it).

`max_tries` Maximum retry attempts.

See Also

`resolve_api_key()`,

Examples

```
## Not run:
# Basic connection (API key from env var)
conn <- kagi_connection()
conn

# Explicit API key
conn2 <- kagi_connection(api_key = "my-key")
```

```
# Lazy API key via keyring
conn3 <- kagi_connection(api_key = function() keyring::key_get("API_kagi"))

## End(Not run)
```

---

kagi_fetch *Fetch Kagi Data into an Endpoint-Structured Project Folder*

Description

High-level helper that runs `kagi_request()` and `kagi_request_parquet()` in sequence and writes outputs into endpoint-scoped project folders.

Usage

```
kagi_fetch(
  connection,
  query,
  project_folder = NULL,
  endpoint = NULL,
  overwrite = FALSE,
  workers = 1,
  limit = NULL,
  verbose = FALSE,
  error_mode = c("stop", "write_dummy")
)
```

Arguments

`connection`	A `kagi_connection()` object.
`query`	A query object of class `kagi_query_*` or a list of query objects.
`project_folder`	Root folder for endpoint-scoped outputs. If `NULL`, a temporary directory is used.
`endpoint`	Optional endpoint override. One of `"search"`, `"enrich_web"`, `"enrich_news"`, `"summarize"`, `"fastgpt"`.
`overwrite`	Logical. If `TRUE`, endpoint output folders are overwritten.
`workers`	Number of workers for list requests.
`limit`	Optional integer limit used for search/enrich request calls.
`verbose`	Logical indicating whether progress messages should be shown.
`error_mode`	Error handling mode passed to `kagi_request()`. One of `"stop"` or `"write_dummy"`.

Details

Folder layout:

- `<project_folder>/<endpoint>/json`
- `<project_folder>/<endpoint>/parquet`

Value

For a single endpoint, normalized parquet path. For mixed endpoint query lists, a named list of normalized parquet paths by endpoint.

Examples

```
## Not run:
conn <- kagi_connection(api_key = function() keyring::key_get("API_kagi"))
q <- query_search("biodiversity", expand = FALSE)

kagi_fetch(
  connection = conn,
  query = q,
  project_folder = "kagi_project"
)

## End(Not run)
```

`kagi_request` *Execute Kagi API requests and save JSON responses*

Description

Execute one or more `kagiPro` query objects against the Kagi API and write raw JSON responses to disk. This function supports search, enrich (web/news), summarize, and FastGPT query classes generated by `query_search()`, `query_enrich_web()`, `query_enrich_news()`, `query_summarize()`, and `query_fastgpt()`.

Usage

```
kagi_request(
  connection,
  query,
  limit = NULL,
  output = NULL,
  overwrite = FALSE,
  append = FALSE,
  workers = 1,
  verbose = FALSE,
  error_mode = c("stop", "write_dummy"),
  metadata_request_args = list()
)
```

Arguments

connection	A kagi_connection() object.
query	A query object of class kagi_query_search, kagi_query_summarize, kagi_query_enrich_web, kagi_query_enrich_news, kagi_query_fastgpt, or a list of such objects.
limit	Optional integer limit used for search and enrich endpoints.
output	Directory where JSON response files are written.
overwrite	Logical. If TRUE, existing output is deleted before writing.
append	Logical. If TRUE, write into an existing output directory instead of deleting it.
workers	Number of parallel workers to use when query is a list.
verbose	Logical indicating whether progress messages should be shown.
error_mode	Error handling mode. "stop" (default) throws on request errors. "write_dummy" writes a fallback JSON payload and returns output.
metadata_request_args	Optional named list persisted in replay metadata (request_args) for each query. Intended for higher-level orchestrators.

Details

If query is a list of query objects, requests are executed in parallel (using workers) and each query is written into a named subdirectory under output.

Files are written as {endpoint}_{page}.json (for example search_1.json). Pagination is handled via meta$next_cursor when provided by the API.

Query replay metadata is written alongside JSON outputs:

- per query folder: _query_meta.json

Value

The normalized path to output.

---

kagi_request_parquet *Convert JSON files to Apache Parquet files*

Description

Convert a directory of JSON files written by kagi_request() into an Apache Parquet dataset. JSON files are processed one-by-one and written as hive-partitioned parquet by query.

Usage

```
kagi_request_parquet(
  input_json = NULL,
  output = NULL,
  add_columns = list(),
  overwrite = FALSE,
  append = FALSE,
  verbose = TRUE,
  delete_input = FALSE
)
```

Arguments

input_json	Directory containing JSON files from kagi_request().
output	output directory for the parquet dataset; default: temporary directory.
add_columns	List of additional fields to be added to the output. They have to be provided as a named list, e.g. list(column_1 = "value_1", column_2 = 2). Only Scalar values are supported.
overwrite	Logical indicating whether to overwrite output.
append	Logical indicating whether to append/update query partitions in an existing output directory without deleting untouched queries.
verbose	Logical indicating whether to print progress information. Defaults to TRUE
delete_input	Determines if the input_json should be deleted afterwards. Defaults to FALSE.

Details

The function uses DuckDB to read the JSON files and to create the Apache Parquet files. It creates an in-memory DuckDB connection, reads each JSON response, and writes endpoint-specific tabular data into the parquet dataset. Files with data = null are skipped.

Output parquet rows include an id column for traceability:

- Search: SEARCH_<hash> from normalized url when available.
- Enrich web: ENRICH_WEB_<hash> from normalized url when available.
- Enrich news: ENRICH_NEWS_<hash> from normalized url when available.
- Summarize: SUMMARIZE_<hash> from request metadata.
- FastGPT: FASTGPT_<hash> from request metadata.

Value

Returns output invisibly if parquet files were written; otherwise NULL.

kagi_update_query *Re-Run a Stored Query by Name and Refresh Parquet*

Description

Update one query dataset by `query_name` using metadata written by `kagi_request()`. The function scans per-query metadata files under `<project_folder>/<endpoint>/json/<query_name>/_query_meta.json`, re-runs all matching query definitions, and refreshes only the touched parquet query partitions.

Usage

```
kagi_update_query(
  connection,
  project_folder,
  query_name,
  workers = 1,
  verbose = FALSE,
  error_mode = c("stop", "write_dummy")
)
```

Arguments

`connection`	A `kagi_connection()` object.
`project_folder`	Root project folder containing endpoint subfolders.
`query_name`	Query name to update (for example `"query_1"` or `"biodiversity_main"`).
`workers`	Number of workers for request execution.
`verbose`	Logical indicating whether progress messages should be shown.
`error_mode`	Error handling mode passed to `kagi_request()`. One of `"stop"` or `"write_dummy"`.

Details

If the same `query_name` exists across multiple endpoints, all matching endpoints are updated.

Value

Named list of normalized parquet output paths by updated endpoint.

Examples

```
## Not run:
kagi_update_query(
  connection = conn,
  project_folder = "kagi_project",
  query_name = "query_1"
)

## End(Not run)
```

`markdown_abstract` *Summarize Markdown into Query-Level Abstract Parquet*

Description

Read markdown files generated for a specific endpoint/query and summarize each record with either OpenAI or Kagi text summarization. The result is written as a single parquet file per query under ‘abstract/‘.

Usage

```
markdown_abstract(
  project_folder,
  endpoint = NULL,
  query_name = NULL,
  workers = 4,
  progress = interactive(),
  verbose = FALSE,
  summarizer_fn = summarize_with_openai,
  model = "gpt-4.1-mini",
  connection = NULL,
  provider_args = list(),
  markdown_root = "markdown",
  abstract_root = "abstract"
)
```

Arguments

`project_folder`	Root project folder containing endpoint subfolders.
`endpoint`	Optional endpoint selector (for example ‘"search"‘ or ‘"enrich_news"‘). If ‘NULL‘, all supported endpoints are considered.
`query_name`	Optional query selector. If ‘NULL‘, all query partitions are considered.
`workers`	Number of parallel workers to use for summarization.
`progress`	Logical indicating whether progress messages should be shown.
`verbose`	Logical indicating whether detailed messages should be shown.
`summarizer_fn`	Function with signature ‘fn(text, model, ...) -> character(1) | NA_character_‘.
`model`	Provider-specific model/engine.
`connection`	Optional [kagi_connection()] object. Used for [summarize_with_kagi()] when not supplied via ‘provider_args‘.
`provider_args`	Optional named list forwarded to ‘summarizer_fn‘.
`markdown_root`	Root folder name containing markdown files.
`abstract_root`	Root folder name for abstract parquet outputs.

Value

Invisibly returns a data frame with columns ‘endpoint‘, ‘id‘, ‘query‘, ‘abstract‘, ‘status‘, ‘error‘.

`open_search_query` *Open a Kagi search in the browser*

Description

Open a Kagi search in the browser

Usage

```
open_search_query(query, session_token = NULL)
```

Arguments

`query`	A full query string (typically from [query_search()]).
`session_token`	Optional Kagi session token for private search (see your Kagi account’s "Session Link").

`query_enrich_news` *Build a Kagi search query string*

Description

Construct one or more query strings for the Kagi Search API by combining free-text terms with structured operators such as `filetype:`, `site:`, `inurl:`, and `intitle:`. Use `kagi_request()` to execute the request and obtain the json replies.

Usage

```
query_enrich_news(
  query,
  filetype = NULL,
  site = NULL,
  inurl = NULL,
  intitle = NULL,
  expand = TRUE
)
```

Arguments

query	Character vector of free-text query terms (required). These can include quoted phrases and boolean operators.
filetype	Optional character vector of file type extensions (e.g. "pdf", "docx"). Each is prefixed with filetype:.
site	Optional character vector of domains (e.g. "example.com", "gov"). Each is prefixed with site:.
inurl	Optional character vector of URL substrings that must be present in the result URL. Each is prefixed with inurl:.
intitle	Optional character vector of terms that must appear in the page title. Each is prefixed with intitle:.
expand	Logical, default TRUE. If TRUE, generate a fully crossed set of queries (Cartesian product of all combinations of query, filetype, site, inurl, and intitle). If FALSE, concatenate the arguments into a single combined query string.

Details

This helper makes it easy to build reproducible, complex queries with structured operators. Use expand = TRUE when you want all possible combinations (useful in systematic search contexts). Use expand = FALSE when you want a single combined query.

Value

A named list containing query strings of class kagi_query_enrich_news, to be used in kagi_request().

See Also

open_search_query(), kagi_request(), kagi_request_parquet(),

Examples

```
## Not run:
# Single combined query
query_search(
  query = "biodiversity",
  filetype = c("pdf", "docx"),
  site = "example.com",
  expand = FALSE
)

# Expanded combinations
query_search(
  query = c("biodiversity", "ecosystem"),
  filetype = c("pdf", "docx"),
  site = c("example.com", "gov"),
  expand = TRUE
)

# Open a generated query manually in browser
```

```
open_search_query(query_search("openalex api", site = "docs.openalex.org")[[1]])

## End(Not run)
```

`query_enrich_web` *Build a Kagi search query string*

Description

Construct one or more query strings for the Kagi Search API by combining free-text terms with structured operators such as `filetype:`, `site:`, `inurl:`, and `intitle:`. Use `kagi_request()` to execute the request and obtain the json replies.

Usage

```
query_enrich_web(
  query,
  filetype = NULL,
  site = NULL,
  inurl = NULL,
  intitle = NULL,
  expand = TRUE
)
```

Arguments

`query`	Character vector of free-text query terms (required). These can include quoted phrases and boolean operators.
`filetype`	Optional character vector of file type extensions (e.g. `"pdf"`, `"docx"`). Each is prefixed with `filetype:`.
`site`	Optional character vector of domains (e.g. `"example.com"`, `"gov"`). Each is prefixed with `site:`.
`inurl`	Optional character vector of URL substrings that must be present in the result URL. Each is prefixed with `inurl:`.
`intitle`	Optional character vector of terms that must appear in the page title. Each is prefixed with `intitle:`.
`expand`	Logical, default `TRUE`. If `TRUE`, generate a fully crossed set of queries (Cartesian product of all combinations of `query`, `filetype`, `site`, `inurl`, and `intitle`). If `FALSE`, concatenate the arguments into a single combined query string.

Details

This helper makes it easy to build reproducible, complex queries with structured operators. Use `expand = TRUE` when you want all possible combinations (useful in systematic search contexts). Use `expand = FALSE` when you want a single combined query.

Value

A named list containing query strings of class `kagi_query_enrich_web`, to be used in `kagi_request()`.

See Also

`open_search_query()`, `kagi_request()`, `kagi_request_parquet()`,

Examples

```
## Not run:
# Single combined query
query_search(
  query = "biodiversity",
  filetype = c("pdf", "docx"),
  site = "example.com",
  expand = FALSE
)

# Expanded combinations
query_search(
  query = c("biodiversity", "ecosystem"),
  filetype = c("pdf", "docx"),
  site = c("example.com", "gov"),
  expand = TRUE
)

# Open a generated query manually in browser
open_search_query(query_search("openalex api", site = "docs.openalex.org")[[1]])

## End(Not run)
```

---

query_fastgpt *Create a FastGPT query payload*

Description

Construct one or more FastGPT query payloads for `POST /fastgpt`. Use `kagi_request()` to execute the request and obtain JSON responses.

Usage

```
query_fastgpt(query, cache = TRUE, web_search = TRUE)
```

Arguments

`query`	Character vector. Query text to answer.
`cache`	Logical. Whether cached responses are allowed. Default: `TRUE`.
`web_search`	Logical. Whether to use web search enrichment. Default: `TRUE`.

Details

According to current Kagi FastGPT API behavior, `web_search = FALSE` is out of service and rejected. This constructor enforces `web_search = TRUE`.

Value

A named list of query objects of class `kagi_query_fastgpt` to be used in `kagi_request()`.

Examples

```
## Not run:
query_fastgpt("Python 3.11")
query_fastgpt(c("Python 3.11", "What is biodiversity?"))

## End(Not run)
```

---

query_search *Build a Kagi search query string*

Description

Construct one or more query strings for the Kagi Search API by combining free-text terms with structured operators such as `filetype:`, `site:`, `inurl:`, and `intitle:`. Queries can either be concatenated into a single string or expanded into a Cartesian product of all combinations.

Usage

```
query_search(
  query,
  filetype = NULL,
  site = NULL,
  inurl = NULL,
  intitle = NULL,
  expand = TRUE,
  open_in_browser = FALSE
)
```

Arguments

`query`	Character vector of free-text query terms (required). These can include quoted phrases and boolean operators.
`filetype`	Optional character vector of file type extensions (e.g. `"pdf"`, `"docx"`). Each is prefixed with `filetype:`.
`site`	Optional character vector of domains (e.g. `"example.com"`, `"gov"`). Each is prefixed with `site:`.

inurl	Optional character vector of URL substrings that must be present in the result URL. Each is prefixed with inurl:.
intitle	Optional character vector of terms that must appear in the page title. Each is prefixed with intitle:.
expand	Logical, default TRUE. If TRUE, generate a fully crossed set of queries (Cartesian product of all combinations of query, filetype, site, inurl, and intitle). If FALSE, concatenate the arguments into a single combined query string.
open_in_browser	Logical, default FALSE. If TRUE, each generated query is immediately opened in the default web browser via open_search_query() for inspection.

Details

This helper makes it easy to build reproducible, complex queries with structured operators. Use expand = TRUE when you want all possible combinations (useful in systematic search contexts). Use expand = FALSE when you want a single combined query.

Value

A named list containing the query strings of type kagi_query_search, to be used in kagi_request().

See Also

open_search_query(), kagi_request(), kagi_request_parquet(),

Examples

```
## Not run:
# Single combined query
query_search(
  query = "biodiversity",
  filetype = c("pdf", "docx"),
  site = "example.com",
  expand = FALSE
)

# Expanded combinations
query_search(
  query = c("biodiversity", "ecosystem"),
  filetype = c("pdf", "docx"),
  site = c("example.com", "gov"),
  expand = TRUE
)

# Immediately open in browser
query_search("openalex api", site = "docs.openalex.org", open_in_browser = TRUE)

## End(Not run)
```

query_summarize *Create a new Kagi summarize request*

Description

Construct a typed S3 object of class `kagi_summarize` that describes a Universal Summarizer request. Use `kagi_request()` to execute the request and obtain the json replies.

Usage

```
query_summarize(
  url = NULL,
  text = NULL,
  engine = NULL,
  summary_type = NULL,
  target_language = NULL,
  cache = TRUE
)
```

Arguments

`url`	Optional character scalar. URL to be summarized. Mutually exclusive with `text`.
`text`	Optional character scalar. Raw text to be summarized. Mutually exclusive with `url`.
`engine`	Character scalar. Summarizer engine (options: `"cecil"`, `"agnes"`, `"muriel"`, `"daphne"`). Default: `"cecil"`.
`summary_type`	Character scalar. Type of summary requested (options: `"summary"`, `"takeaway"`). Default: `"summary"`.
`target_language`	Character scalar. Target language ISO code. Supported codes: `"EN"`, `"BG"`, `"CS"`, `"DA"`, `"DE"`, `"EL"`, `"ES"`, `"ET"`, `"FI"`, `"FR"`, `"HU"`, `"ID"`, `"IT"`, `"JA"`, `"KO"`, `"LT"`, `"LV"`, `"NB"`, `"NL"`, `"PL"`, `"PT"`, `"RO"`, `"RU"`, `"SK"`, `"SL"`, `"SV"`, `"TR"`, `"UK"`, `"ZH"`, `"ZH-HANT"`. Default: `"EN"`.
`cache`	Logical. Whether to allow API-side caching.

Value

A named list of `kagi_query_summarize` objects to be passed to `kagi_request()`.

Examples

```
## Not run:
req <- query_summarize(text = "Lorem ipsum")
req

## End(Not run)
```

`read_corpus` *Read a kagiPro Parquet Corpus*

Description

Modelled on ‘openalexPro::read_corpus()‘ with an additional ‘abstracts‘ switch. By default this opens an Arrow dataset from a parquet directory. When ‘return_data = TRUE‘, the result is collected into memory.

Usage

```
read_corpus(
  project_folder,
  endpoint,
  corpus = "parquet",
  return_data = FALSE,
  abstracts = FALSE,
  silent = FALSE
)
```

Arguments

`project_folder`	Root project folder.
`endpoint`	Endpoint folder name under ‘project_folder‘.
`corpus`	Folder name under ‘project_folder/endpoint‘ to read as parquet corpus. Defaults to ‘"parquet"‘.
`return_data`	Logical; if ‘TRUE‘, collect and return in-memory data.
`abstracts`	Logical; if ‘TRUE‘, link sibling abstract data by ‘id‘ and ‘query‘.
`silent`	Logical; if ‘TRUE‘, suppress informative messages.

Details

If ‘abstracts = TRUE‘, abstract data is read from the sibling ‘abstract‘ folder and left-joined by ‘id‘ + ‘query‘. If no abstract files are present, an ‘abstract‘ column filled with ‘NA‘ is added.

Value

An Arrow dataset/query when ‘return_data = FALSE‘, otherwise a data frame/tibble.

summarize_with_kagi *Summarize Text via Kagi Summarize Endpoint*

Description

Summarize Text via Kagi Summarize Endpoint

Usage

```
summarize_with_kagi(
  text,
  model = "cecil",
  connection = NULL,
  api_key = NULL,
  base_url = NULL,
  summary_type = "summary",
  target_language = "EN",
  cache = TRUE,
  retry_max_tries = 5
)
```

Arguments

text	Plain text to summarize.
model	Kagi summarize engine (‘"cecil"‘, ‘"agnes"‘, ‘"muriel"‘, ‘"daphne"‘).
connection	Optional [kagi_connection()] object.
api_key	Optional Kagi API key override.
base_url	Optional Kagi API base URL override.
summary_type	Summarize mode (‘"summary"‘ or ‘"takeaway"‘).
target_language	Target language code.
cache	Cache flag forwarded to Kagi summarize endpoint.
retry_max_tries	Maximum number of HTTP retry attempts passed to [httr2::req_retry()].

Value

A single summary string (or ‘NA_character_‘).

summarize_with_openai *Summarize Text via OpenAI Chat Completions*

Description

Summarize Text via OpenAI Chat Completions

Usage

```
summarize_with_openai(
  text,
  model = "gpt-4.1-mini",
  api_key = Sys.getenv("API_openai", ""),
  base_url = "https://api.openai.com/v1",
 system_prompt = "Summarize input text in 4-6 concise sentences for literature review.",
  retry_max_tries = 5
)
```

Arguments

text	Plain text to summarize.
model	OpenAI model name.
api_key	OpenAI API key. Defaults to 'API_openai'.
base_url	OpenAI API base URL.
system_prompt	Prompt used to guide summarization behavior.
retry_max_tries	Maximum number of HTTP retry attempts passed to [httr2::req_retry()].

Value

A single summary string (or 'NA_character_').

Index