Package: VectrixDB (via r-universe)

May 22, 2026

Type Package

Title Lightweight Vector Database with Embedded Machine Learning Models

Version 1.1.2

Author Kwadwo Daddy Nyame Owusu Boakye [aut, cre]

Maintainer Kwadwo Daddy Nyame Owusu Boakye <kwadwo.owusuboakye@outlook.com>

Description A lightweight vector database for text retrieval in R with embedded machine learning models and no external API (Application Programming Interface) keys. Supports dense and hybrid search, optional HNSW (Hierarchical Navigable Small World) approximate nearest-neighbor indexing, faceted filters with ACL (Access Control List) metadata, command-line tools, and a local dashboard built with 'shiny'. The HNSW method is described by Malkov and Yashunin (2018) <doi:10.1109/TPAMI.2018.2889473>.

License Apache License (>= 2)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports R6, jsonlite, digest, Matrix, text2vec (>= 0.6.0), stopwords, tools, utils, stats, RSQLite, DBI

Suggests testthat (>= 3.0.0), knitr, rmarkdown, shiny, plumber, pkgdown, RcppAnnoy, reticulate, rappdirs

VignetteBuilder knitr

Config/testthat/edition 3

URL https://knowusuboaky.github.io/vectrixdb-r/, https://github.com/knowusuboaky/vectrixdb-r

BugReports https://github.com/knowusuboaky/vectrixdb-r/issues

Config/pak/sysreqs libicu-dev

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

Date/Publication 2026-02-18 19:00:21 UTC

RemoteUrl https://github.com/knowusuboaky/vectrixdb-r

RemoteRef HEAD

RemoteSha 574f3a48e14174321134c9cd44d76d324406cc59

Contents

`acl_config_from_list` *Create ACL Config from List*

Description

Create ACLConfig from list of ACL strings

Usage

```
acl_config_from_list(acl_list)
```

Arguments

`acl_list`	Character vector of ACL strings

Value

ACLConfig object

ACLConfig *ACL Configuration*

Description

ACL configuration for a document or collection

Public fields

`read_principals` Who can read

`deny_principals` Who cannot read (takes precedence)

`is_public` Is public access allowed

Methods

Public methods:

- `ACLConfig$new()`
- `ACLConfig$clone()`

Method `new()`**:** Create a new ACLConfig

Usage:

```
ACLConfig$new(
  read_principals = list(),
  deny_principals = list(),
  is_public = FALSE
)
```

Arguments:

`read_principals` List of ACLPrincipal objects

`deny_principals` List of ACLPrincipal objects

`is_public` Logical

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ACLConfig$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

ACLFilter *ACL Filter*

Description

Access Control List filter for security-aware search

Public fields

`acl_field` Metadata field containing ACLs

Methods

Public methods:

- `ACLFilter$new()`
- `ACLFilter$filter()`
- `ACLFilter$add_acl()`
- `ACLFilter$create_filter_condition()`
- `ACLFilter$clone()`

Method `new()`**:** Create a new ACLFilter

Usage:

```
ACLFilter$new(acl_field = "_acl")
```

Arguments:

`acl_field` Field name for ACLs (default: "_acl")

Method `filter()`**:** Filter documents based on user's ACL principals

Usage:

```
ACLFilter$filter(documents, user_principals, default_allow = FALSE)
```

Arguments:

`documents` List of documents with metadata
`user_principals` Character vector or list of ACLPrincipal
`default_allow` Allow if no ACL defined (default: FALSE)

Returns: Filtered documents

Method `add_acl()`**:** Add ACL to document metadata

Usage:

```
ACLFilter$add_acl(metadata, principals)
```

Arguments:

`metadata` Document metadata
`principals` Character vector of principal strings

Returns: Updated metadata

Method `create_filter_condition()`**:** Create ACL filter condition for query

Usage:

```
ACLFilter$create_filter_condition(user_principals)
```

Arguments:

`user_principals` Character vector of principals

Returns: Filter condition list

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ACLFilter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
acl_filter <- ACLFilter$new()
filtered <- acl_filter$filter(
  documents = results,
  user_principals = c("user:alice", "group:engineering")
)

## End(Not run)
```

`ACLOperator` *ACL Operator Types*

Description

ACL matching operators

Usage

```
ACLOperator
```

Format

An object of class `list` of length 5.

`ACLPrincipal` *ACL Principal*

Description

An ACL principal (user, group, or role)

Public fields

`type` Principal type

`value` Principal value

Methods

Public methods:

- `ACLPrincipal$new()`
- `ACLPrincipal$matches()`
- `ACLPrincipal$to_string()`
- `ACLPrincipal$clone()`

Method `new()`**:** Create a new ACLPrincipal

Usage:

```
ACLPrincipal$new(type, value)
```

Arguments:

`type` Principal type (user, group, role)

`value` Principal value

Method `matches()`**:** Check if this principal matches another

Usage:

```
ACLPrincipal$matches(other)
```

Arguments:

`other` Another ACLPrincipal

Returns: Logical

Method `to_string()`**:** Convert to string

Usage:

```
ACLPrincipal$to_string()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ACLPrincipal$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

advanced_search *VectrixDB Advanced Search Features*

Description

Enterprise-grade search capabilities:

- Faceted search with aggregations
- ACL/Security filtering
- Text analyzers (stemming, synonyms, stopwords)

AdvancedReranker *Advanced Reranker with Learned Weights*

Description

Combines multiple signals for better reranking:

- Semantic similarity (word vectors)
- BM25/keyword overlap
- Query coverage
- Position bias
- Length normalization

Public fields

`weights` Feature weights

Methods

Public methods:

- `AdvancedReranker$new()`
- `AdvancedReranker$set_embedder()`
- `AdvancedReranker$rerank()`
- `AdvancedReranker$learn_weights()`
- `AdvancedReranker$clone()`

Method `new()`**:** Create a new AdvancedReranker

Usage:

```
AdvancedReranker$new(
  semantic_weight = 0.4,
  bm25_weight = 0.3,
  coverage_weight = 0.2,
  position_weight = 0.1,
  sentence_embedder = NULL
)
```

Arguments:

`semantic_weight` Weight for semantic similarity (0-1)

`bm25_weight` Weight for BM25 score (0-1)

`coverage_weight` Weight for query term coverage (0-1)

`position_weight` Weight for position bias (0-1)

`sentence_embedder` Optional SentenceEmbedder for semantic scoring

Method `set_embedder()`**:** Set sentence embedder

Usage:

```
AdvancedReranker$set_embedder(embedder)
```

Arguments:

`embedder` SentenceEmbedder object

Method `rerank()`**:** Rerank results

Usage:

```
AdvancedReranker$rerank(
  query,
  query_vector = NULL,
  results,
  doc_vectors = NULL,
  limit = 10
)
```

Arguments:

`query` Query text

`query_vector` Query embedding vector

`results` List of result objects with id, text, score

`doc_vectors` Matrix of document vectors (optional)

`limit` Number of results to return

Returns: Reranked list of results

Method `learn_weights()`**:** Learn optimal weights from relevance judgments

Usage:

```
AdvancedReranker$learn_weights(
  queries,
  results_list,
  relevance_list,
  iterations = 100
)
```

Arguments:

`queries` Character vector of queries

`results_list` List of result lists (one per query)

`relevance_list` List of relevance scores (1=relevant, 0=not)

`iterations` Number of optimization iterations

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
AdvancedReranker$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`AnalyzerChain` *Analyzer Chain*

Description

Chain multiple analyzers together

Public fields

`analyzers` List of TextAnalyzer objects

Methods

Public methods:

- `AnalyzerChain$new()`
- `AnalyzerChain$analyze()`
- `AnalyzerChain$clone()`

Method `new()`**:** Create a new AnalyzerChain

Usage:

```
AnalyzerChain$new(analyzers)
```

Arguments:

`analyzers` List of TextAnalyzer objects

Method `analyze()`**:** Run text through all analyzers

Usage:

```
AnalyzerChain$analyze(text)
```

Arguments:

`text` Input text

Returns: Character vector of tokens

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
AnalyzerChain$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`BaseCache` *Base Cache*

Description

Abstract base class for cache backends

Public fields

`config` Cache configuration

`stats` Cache statistics

Methods

Public methods:

- `BaseCache$new()`
- `BaseCache$get()`
- `BaseCache$set()`
- `BaseCache$delete()`
- `BaseCache$exists()`
- `BaseCache$clear()`
- `BaseCache$size()`
- `BaseCache$get_many()`
- `BaseCache$set_many()`
- `BaseCache$delete_many()`
- `BaseCache$make_key()`
- `BaseCache$clone()`

Method `new()`**:** Create a new cache

Usage:

```
BaseCache$new(config = NULL)
```

Arguments:

`config` CacheConfig object

Method `get()`**:** Get a value from cache

Usage:

```
BaseCache$get(key)
```

Arguments:

`key` Cache key

Returns: Cached value or NULL

Method `set()`**:** Set a value in cache

Usage:

```
BaseCache$set(key, value, ttl = NULL)
```

Arguments:

`key` Cache key

`value` Value to cache

`ttl` Time to live (optional)

Method `delete()`**:** Delete a key from cache

Usage:

```
BaseCache$delete(key)
```

Arguments:

`key` Cache key

Returns: Logical success

Method `exists()`**:** Check if key exists

Usage:

```
BaseCache$exists(key)
```

Arguments:

`key` Cache key

Returns: Logical

Method `clear()`**:** Clear all cache entries

Usage:

```
BaseCache$clear()
```

Method `size()`**:** Get cache size

Usage:

```
BaseCache$size()
```

Returns: Integer count

Method `get_many()`**:** Get multiple values

Usage:

```
BaseCache$get_many(keys)
```

Arguments:

`keys` Character vector of keys

Returns: Named list of values

Method `set_many()`**:** Set multiple values

Usage:

```
BaseCache$set_many(items, ttl = NULL)
```

Arguments:

`items` Named list of values

`ttl` Time to live

Method `delete_many()`**:** Delete multiple keys

Usage:

```
BaseCache$delete_many(keys)
```

Arguments:

`keys` Character vector of keys

Returns: Integer count of deleted keys

Method `make_key()`**:** Make a prefixed key

Usage:

```
BaseCache$make_key(key)
```

Arguments:

`key` Raw key

Returns: Prefixed key

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
BaseCache$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`cache` *VectrixDB Cache Layer*

Description

High-performance caching for low latency

Supports multiple cache backends:

- InMemory LRU: Ultra-fast, limited by RAM
- File-based: Persistent cache using RDS files

cache_config_from_env *Create Cache Config from Environment*

Description

Create config from environment variables

Usage

```
cache_config_from_env()
```

Value

CacheConfig object

CacheBackend *Cache Backend Types*

Description

Available cache backends

Usage

```
CacheBackend
```

Format

An object of class `list` of length 3.

CacheConfig *Cache Configuration*

Description

Configuration for cache layer

Public fields

`backend` Cache backend type

`memory_max_size` Max items in memory

`memory_ttl_seconds` Default TTL in seconds

`file_cache_dir` Directory for file cache

`file_ttl_seconds` File cache TTL

`prefix` Cache key prefix

`compression` Use compression

Methods

Public methods:

- CacheConfig$new()
- CacheConfig$clone()

Method `new()`**:** Create a new CacheConfig

Usage:

```
CacheConfig$new(
  backend = "memory",
  memory_max_size = 10000,
  memory_ttl_seconds = 3600,
  file_cache_dir = NULL,
  file_ttl_seconds = 86400,
  prefix = "vectrix:",
  compression = TRUE
)
```

Arguments:

`backend` Backend type

`memory_max_size` Max memory items

`memory_ttl_seconds` Memory TTL

`file_cache_dir` File cache directory

`file_ttl_seconds` File TTL

`prefix` Key prefix

`compression` Use compression

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CacheConfig$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CacheEntry` *Cache Entry*

Description

A cached entry with metadata

Public fields

`value` Cached value

`created_at` Creation timestamp

`ttl` Time to live in seconds

`hits` Number of hits

Methods

Public methods:

- `CacheEntry$new()`
- `CacheEntry$is_expired()`
- `CacheEntry$clone()`

Method `new()`**:** Create a new CacheEntry

Usage:

```
CacheEntry$new(value, ttl)
```

Arguments:

`value` The value to cache

`ttl` Time to live in seconds

Method `is_expired()`**:** Check if entry is expired

Usage:

```
CacheEntry$is_expired()
```

Returns: Logical

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CacheEntry$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CacheStats` *Cache Statistics*

Description

Cache statistics for monitoring

Public fields

`hits` Cache hits

`misses` Cache misses

`sets` Cache sets

`deletes` Cache deletes

`evictions` Cache evictions

Methods

Public methods:

- CacheStats$record_hit()
- CacheStats$record_miss()
- CacheStats$record_set()
- CacheStats$record_delete()
- CacheStats$record_eviction()
- CacheStats$hit_rate()
- CacheStats$to_list()
- CacheStats$reset()
- CacheStats$clone()

Method `record_hit()`**:** Record a cache hit

Usage:

```
CacheStats$record_hit()
```

Method `record_miss()`**:** Record a cache miss

Usage:

```
CacheStats$record_miss()
```

Method `record_set()`**:** Record a cache set

Usage:

```
CacheStats$record_set()
```

Method `record_delete()`**:** Record a cache delete

Usage:

```
CacheStats$record_delete()
```

Method `record_eviction()`**:** Record a cache eviction

Usage:

```
CacheStats$record_eviction()
```

Method `hit_rate()`**:** Get hit rate

Usage:

```
CacheStats$hit_rate()
```

Returns: Numeric hit rate

Method `to_list()`**:** Convert to list

Usage:

```
CacheStats$to_list()
```

Returns: List representation

Method `reset()`**:** Reset statistics

Usage:

```
CacheStats$reset()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CacheStats$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`cli` *VectrixDB Command Line Interface*

Description

CLI tools for VectrixDB operations

Provides command-line style functions for:

- Creating and managing collections
- Adding and searching documents
- Exporting and importing data
- Database statistics and info

`CLIConfig` *CLI Configuration*

Description

Configuration for CLI behavior

Public fields

`verbose` Print verbose output

`color` Use colored output

`data_dir` Default data directory

Methods

Public methods:

- CLIConfig$new()
- CLIConfig$clone()

Method `new()`**:** Create CLI config

Usage:

```
CLIConfig$new(verbose = TRUE, color = TRUE, data_dir = NULL)
```

Arguments:

`verbose` Verbose output

`color` Colored output

`data_dir` Data directory

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CLIConfig$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Collection` *Collection Class*

Description

Vector collection with indexing and search

Public fields

`name` Collection name

`dimension` Vector dimension

`metric` Distance metric

`language` Language setting ("en" or "ml")

Methods

Public methods:

- Collection$new()
- Collection$add()
- Collection$search()
- Collection$keyword_search()
- Collection$hybrid_search()
- Collection$get()

- Collection$delete()
- Collection$count()
- Collection$clear()
- Collection$clone()

Method `new()`**:** Create a new Collection

Usage:

```
Collection$new(
  name,
  dimension,
  metric = "cosine",
  storage = NULL,
  language = "en"
)
```

Arguments:

`name` Collection name

`dimension` Vector dimension

`metric` Distance metric

`storage` Storage backend

`language` Language behavior ("en" = ASCII-focused, "ml" = Unicode-aware)

Method `add()`**:** Add documents to collection

Usage:

```
Collection$add(ids, vectors, metadata = NULL, texts = NULL)
```

Arguments:

`ids` Document IDs

`vectors` Matrix of vectors

`metadata` List of metadata

`texts` Character vector of texts

Method `search()`**:** Search collection

Usage:

```
Collection$search(query, limit = 10, filter = NULL, include_vectors = FALSE)
```

Arguments:

`query` Query vector

`limit` Number of results

`filter` Metadata filter

`include_vectors` Include vectors in results

Returns: Results object

Method `keyword_search()`**:** Keyword search

Usage:

```
Collection$keyword_search(query_text, limit = 10, filter = NULL)
```

Arguments:

`query_text` Query text

`limit` Number of results

`filter` Metadata filter

Returns: Results object

Method `hybrid_search()`**:** Hybrid search (dense + sparse)

Usage:

```
Collection$hybrid_search(
  query,
  query_text,
  limit = 10,
  vector_weight = 0.5,
  text_weight = 0.5,
  filter = NULL,
  include_vectors = FALSE,
  rrf_k = 60,
  prefetch_multiplier = 10
)
```

Arguments:

`query` Query vector

`query_text` Query text

`limit` Number of results

`vector_weight` Weight for vector search

`text_weight` Weight for text search

`filter` Metadata filter

`include_vectors` Include vectors in results

`rrf_k` RRF constant

`prefetch_multiplier` Prefetch multiplier

Returns: Results object

Method `get()`**:** Get documents by ID

Usage:

```
Collection$get(ids)
```

Arguments:

`ids` Document IDs

Returns: List of results

Method `delete()`**:** Delete documents by ID

Usage:

```
Collection$delete(ids)
```

Arguments:

`ids` Document IDs to delete

Method `count()`**:** Get document count

Usage:

```
Collection$count()
```

Returns: Integer count

Method `clear()`**:** Clear collection

Usage:

```
Collection$clear()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Collection$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Community` *Community*

Description

A community of entities

Public fields

`id` Community ID

`level` Hierarchy level

`entity_ids` Member entity IDs

`summary` Community summary

`parent_id` Parent community ID

`child_ids` Child community IDs

Methods

Public methods:

- `Community$new()`
- `Community$size()`
- `Community$clone()`

Method `new()`**:** Create a new Community

Usage:

```
Community$new(
  id,
  level = 0,
  entity_ids = character(0),
  summary = NULL,
  parent_id = NULL
)
```

Arguments:

`id` ID

`level` Level

`entity_ids` Members

`summary` Summary

`parent_id` Parent

Method `size()`**:** Get size

Usage:

```
Community$size()
```

Returns: Integer

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Community$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CommunityDetector` *Simple Community Detector*

Description

Detects communities using connected components

Public fields

`min_size` Minimum community size

`max_levels` Maximum hierarchy levels

Methods

Public methods:

- `CommunityDetector$new()`
- `CommunityDetector$detect()`
- `CommunityDetector$clone()`

Method `new()`**:** Create a new CommunityDetector

Usage:

```
CommunityDetector$new(min_size = 5, max_levels = 3)
```

Arguments:

`min_size` Min size

`max_levels` Max levels

Method `detect()`**:** Detect communities in graph

Usage:

```
CommunityDetector$detect(graph)
```

Arguments:

`graph` KnowledgeGraph object

Returns: List of Community objects

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CommunityDetector$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`create_cache` *Create Cache*

Description

Factory function to create cache backend

Usage

```
create_cache(config = NULL)
```

Arguments

`config`	CacheConfig object or NULL for defaults

Value

Cache object

create_default_graphrag_config *Create Default GraphRAG Config*

Description

Create default config with regex extractor

Usage

```
create_default_graphrag_config(...)
```

Arguments

...	Additional options

Value

GraphRAGConfig object

create_hnsw_index *Create HNSW Index*

Description

Factory function to create HNSW index

Usage

```
create_hnsw_index(dimension, metric = "angular", n_trees = 50)
```

Arguments

dimension	Vector dimension
metric	Distance metric
n_trees	Number of trees

Value

HNSWIndex object

create_pipeline *Create GraphRAG Pipeline*

Description

Factory function for GraphRAGPipeline

Usage

```
create_pipeline(config = NULL)
```

Arguments

config	GraphRAGConfig (optional)

Value

GraphRAGPipeline object

create_sentence_embedder *Create a sentence embedder with automatic download*

Description

Convenience function to create a SentenceEmbedder with GloVe vectors

Usage

```
create_sentence_embedder(model = "glove-100", use_idf = TRUE)
```

Arguments

model	Model name (default: "glove-100")
use_idf	Use IDF weighting

Value

SentenceEmbedder object

Examples

```
## Not run:
# Downloads GloVe if not present
embedder <- create_sentence_embedder("glove-100")

# Embed texts
vectors <- embedder$embed(c("Hello world", "Machine learning is cool"))

## End(Not run)
```

`create_vector_cache` *Create Vector Cache*

Description

Create a VectorCache with specified backend

Usage

```
create_vector_cache(backend = "memory", ...)
```

Arguments

`backend`	Backend type: "memory", "file", or "none"
`...`	Additional config options

Value

VectorCache object

`DenseEmbedder` *Dense Embedder using word2vec or GloVe*

Description

Generates dense vector embeddings using pre-trained word vectors

Public fields

`dimension` Embedding dimension

`model_type` Type of model being used

`language` Language setting ("en" or "ml")

Methods

Public methods:

- DenseEmbedder$new()
- DenseEmbedder$set_sentence_embedder()
- DenseEmbedder$embed()
- DenseEmbedder$fit()
- DenseEmbedder$clone()

Method `new()`**:** Create a new DenseEmbedder

Usage:

```
DenseEmbedder$new(
  dimension = 100,
  model_path = NULL,
  model_type = "tfidf",
  sentence_embedder = NULL,
  auto_download = FALSE,
  language = "en"
)
```

Arguments:

`dimension` Vector dimension (default: 100 for word2vec, 50/100/200/300 for GloVe)

`model_path` Optional path to pre-trained model file

`model_type` Type: "word2vec", "glove", "glove-pretrained", or "tfidf"

`sentence_embedder` Optional SentenceEmbedder object to use

`auto_download` Auto-download GloVe vectors if model_type is glove-pretrained

`language` Language behavior ("en" = ASCII-focused, "ml" = Unicode-aware)

Method `set_sentence_embedder()`**:** Set a SentenceEmbedder to use for embeddings

Usage:

```
DenseEmbedder$set_sentence_embedder(embedder)
```

Arguments:

`embedder` SentenceEmbedder object

Method `embed()`**:** Embed texts to vectors

Usage:

```
DenseEmbedder$embed(texts)
```

Arguments:

`texts` Character vector of texts

Returns: Matrix of embeddings (rows are documents)

Method `fit()`**:** Train embedder on corpus (for TF-IDF)

Usage:

```
DenseEmbedder$fit(texts)
```

Arguments:

`texts` Character vector of training texts

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DenseEmbedder$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`DistanceMetric` *Distance Metric Enumeration*

Description

Available distance metrics for vector comparison

Usage

```
DistanceMetric
```

Format

An object of class `list` of length 4.

`DocumentChunker` *Document Chunker*

Description

Splits documents into text units

Public fields

`chunk_size` Target chunk size

`chunk_overlap` Overlap size

`by_sentence` Preserve sentences

Methods

Public methods:

- DocumentChunker$new()
- DocumentChunker$chunk()
- DocumentChunker$clone()

Method `new()`**:** Create a new DocumentChunker

Usage:

```
DocumentChunker$new(chunk_size = 1200, chunk_overlap = 100, by_sentence = TRUE)
```

Arguments:

`chunk_size` Target size

`chunk_overlap` Overlap

`by_sentence` Preserve sentences

Method `chunk()`**:** Chunk a document

Usage:

```
DocumentChunker$chunk(text, document_id = NULL)
```

Arguments:

`text` Document text

`document_id` Document ID

Returns: List of TextUnit objects

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DocumentChunker$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`download_vectors` *Download pre-trained word vectors*

Description

Download GloVe or other pre-trained word vectors

Usage

```
download_vectors(model = "glove-50", dest_dir = NULL)
```

Arguments

`model`	Model name: "glove-50", "glove-100", "glove-200", "glove-300"
`dest_dir`	Destination directory

Value

Path to downloaded model

---

download_word_vectors *Download pre-trained word vectors*

Description

Downloads GloVe or fastText word vectors

Usage

```
download_word_vectors(model = "glove-100", dest_dir = NULL, overwrite = FALSE)
```

Arguments

model	Model to download: "glove-50", "glove-100", "glove-200", "glove-300", "glove-twitter-25", "glove-twitter-50", "glove-twitter-100", "glove-twitter-200"
dest_dir	Destination directory (default: user cache)
overwrite	Overwrite existing files

Value

Path to the downloaded vectors file

Examples

```
## Not run:
# Download 100-dimensional GloVe vectors (~130MB)
path <- download_word_vectors("glove-100")

# Use with Vectrix
db <- Vectrix$new("docs", model = "glove", model_path = path)

## End(Not run)
```

---

embedders *VectrixDB Embedders (Pure R Implementation)*

Description

Embedding models for text vectorization using R-native packages

ENGLISH_STOPWORDS *English Stopwords*

Description

Common English stopwords

Usage

```
ENGLISH_STOPWORDS
```

Format

An object of class `character` of length 45.

EnhancedSearchResults *Enhanced Search Results*

Description

Search results with enterprise features

Public fields

`results` List of result items

`facets` Named list of FacetResult objects

`total_count` Total results before filtering

`filtered_count` Results after ACL filtering

`query_time_ms` Query time in milliseconds

`rerank_time_ms` Rerank time in milliseconds

`facet_time_ms` Facet time in milliseconds

Methods

Public methods:

- `EnhancedSearchResults$new()`
- `EnhancedSearchResults$to_list()`
- `EnhancedSearchResults$clone()`

Method `new()`**:** Create new EnhancedSearchResults

Usage:

```
EnhancedSearchResults$new(
  results,
  facets = list(),
  total_count = 0,
  filtered_count = 0,
  query_time_ms = 0,
  rerank_time_ms = 0,
  facet_time_ms = 0
)
```

Arguments:

`results` List of results

`facets` Named list of FacetResult

`total_count` Total count

`filtered_count` Filtered count

`query_time_ms` Query time

`rerank_time_ms` Rerank time

`facet_time_ms` Facet time

Method `to_list()`**:** Convert to list

Usage:

```
EnhancedSearchResults$to_list()
```

Returns: List representation

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
EnhancedSearchResults$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Entity` *Entity*

Description

An extracted entity

Public fields

`id` Unique identifier

`name` Entity name

`type` Entity type

`description` Description

`source_chunks` Source chunk IDs

`embedding` Vector embedding

`metadata` Additional metadata

Methods

Public methods:

- Entity$new()
- Entity$to_list()
- Entity$clone()

Method `new()`**:** Create a new Entity

Usage:

```
Entity$new(
  id = NULL,
  name,
  type,
  description = NULL,
  source_chunks = NULL,
  embedding = NULL,
  metadata = NULL
)
```

Arguments:

`id` Unique ID

`name` Name

`type` Type

`description` Description

`source_chunks` Sources

`embedding` Vector

`metadata` Metadata

Method `to_list()`**:** Convert to list

Usage:

```
Entity$to_list()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Entity$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`ExtractionResult` *Extraction Result*

Description

Result of entity extraction

Public fields

`entities` List of Entity objects

`relationships` List of Relationship objects

`source_chunk` Source chunk ID

Methods

Public methods:

- `ExtractionResult$new()`
- `ExtractionResult$clone()`

Method `new()`**:** Create new ExtractionResult

Usage:

```
ExtractionResult$new(
  entities = list(),
  relationships = list(),
  source_chunk = NULL
)
```

Arguments:

`entities` Entities

`relationships` Relationships

`source_chunk` Source

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ExtractionResult$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

ExtractorType *Extractor Types*

Description

Extractor Types

Usage

```
ExtractorType
```

Format

An object of class `list` of length 4.

FacetAggregator *Facet Aggregator*

Description

Faceted search aggregator for computing aggregations/counts

Methods

Public methods:

- `FacetAggregator$aggregate()`
- `FacetAggregator$to_list()`
- `FacetAggregator$clone()`

Method `aggregate()`**:** Aggregate facet values from documents

Usage:

```
FacetAggregator$aggregate(documents, facet_configs)
```

Arguments:

`documents` List of documents with metadata

`facet_configs` List of field names or FacetConfig objects

Returns: Named list mapping field names to FacetResult

Method `to_list()`**:** Convert facet results to list format

Usage:

```
FacetAggregator$to_list(facet_results)
```

Arguments:

`facet_results` Named list of FacetResult objects

Returns: List format suitable for JSON

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FacetAggregator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
aggregator <- FacetAggregator$new()
facets <- aggregator$aggregate(
  documents = list(
    list(category = "tech", author = "Alice"),
    list(category = "science", author = "Bob")
  ),
  facet_fields = c("category", "author")
)

## End(Not run)
```

`FacetConfig` *Facet Configuration*

Description

Configuration for a facet field

Public fields

`field` Field name to facet on

`limit` Max values to return

`min_count` Minimum count to include

`sort_by` Sort by "count" or "value"

`include_zero` Include zero-count values

Methods

Public methods:

- `FacetConfig$new()`
- `FacetConfig$clone()`

Method `new()`**:** Create a new FacetConfig

Usage:

```
FacetConfig$new(
  field,
  limit = 10,
  min_count = 1,
  sort_by = "count",
  include_zero = FALSE
)
```

Arguments:

`field` Field name

`limit` Max values (default: 10)

`min_count` Min count (default: 1)

`sort_by` Sort method (default: "count")

`include_zero` Include zeros (default: FALSE)

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FacetConfig$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`FacetResult` *Facet Result*

Description

Result of facet aggregation

Public fields

`field` Field name

`values` List of FacetValue objects

`total_count` Total count

`other_count` Count of values not in top-N

Methods

Public methods:

- `FacetResult$new()`
- `FacetResult$to_list()`
- `FacetResult$clone()`

Method `new()`**:** Create a new FacetResult

Usage:

```
FacetResult$new(field, values, total_count, other_count = 0)
```

Arguments:

`field` Field name

`values` List of FacetValue objects

`total_count` Total count

`other_count` Other count

Method `to_list()`**:** Convert to list

Usage:

```
FacetResult$to_list()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FacetResult$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`FacetValue` *Facet Value*

Description

A single facet value with count

Public fields

`value` The facet value

`count` Number of occurrences

Methods

Public methods:

- `FacetValue$new()`
- `FacetValue$clone()`

Method `new()`**:** Create a new FacetValue

Usage:

```
FacetValue$new(value, count)
```

Arguments:

`value` The value

`count` The count

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FacetValue$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

FileCache *File Cache*

Description

File-based persistent cache using RDS files

Super class

VectrixDB::BaseCache -> FileCache

Methods

Public methods:

- FileCache$new()
- FileCache$get()
- FileCache$set()
- FileCache$delete()
- FileCache$exists()
- FileCache$clear()
- FileCache$size()
- FileCache$cleanup_expired()
- FileCache$clone()

Method `new()`**:** Create a new FileCache

Usage:

```
FileCache$new(config = NULL)
```

Arguments:

`config` CacheConfig object

Method `get()`**:** Get value from cache

Usage:

```
FileCache$get(key)
```

Arguments:

`key` Cache key

Returns: Value or NULL

Method `set()`**:** Set value in cache

Usage:

```
FileCache$set(key, value, ttl = NULL)
```

Arguments:

`key` Cache key

`value` Value to cache

`ttl` Time to live

Method `delete()`**:** Delete key from cache

Usage:

```
FileCache$delete(key)
```

Arguments:

`key` Cache key

Returns: Logical success

Method `exists()`**:** Check if key exists

Usage:

```
FileCache$exists(key)
```

Arguments:

`key` Cache key

Returns: Logical

Method `clear()`**:** Clear cache

Usage:

```
FileCache$clear()
```

Method `size()`**:** Get cache size

Usage:

```
FileCache$size()
```

Returns: Integer

Method `cleanup_expired()`**:** Cleanup expired entries

Usage:

```
FileCache$cleanup_expired()
```

Returns: Integer count removed

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FileCache$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Filter` *Filter Class for Metadata Filtering*

Description

Build metadata filters for search queries

Public fields

`conditions` List of filter conditions

Methods

Public methods:

- `Filter$new()`
- `Filter$eq()`
- `Filter$ne()`
- `Filter$gt()`
- `Filter$lt()`
- `Filter$in_list()`
- `Filter$to_list()`
- `Filter$clone()`

Method `new()`**:** Create a new Filter

Usage:

```
Filter$new(...)
```

Arguments:

`...` Named filter conditions

Method `eq()`**:** Add equality condition

Usage:

```
Filter$eq(field, value)
```

Arguments:

`field` Field name

`value` Value to match

Returns: Self for chaining

Method `ne()`**:** Add not-equal condition

Usage:

```
Filter$ne(field, value)
```

Arguments:

`field` Field name

`value` Value to exclude

Returns: Self for chaining

Method `gt()`**:** Add greater-than condition

Usage:

```
Filter$gt(field, value)
```

Arguments:

`field` Field name

`value` Threshold value

Returns: Self for chaining

Method `lt()`**:** Add less-than condition

Usage:

```
Filter$lt(field, value)
```

Arguments:

`field` Field name

`value` Threshold value

Returns: Self for chaining

Method `in_list()`**:** Add in-list condition

Usage:

```
Filter$in_list(field, values)
```

Arguments:

`field` Field name

`values` Vector of values

Returns: Self for chaining

Method `to_list()`**:** Convert to list for API

Usage:

```
Filter$to_list()
```

Returns: List representation

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Filter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`GlobalSearcher` *Global Searcher*

Description

Community-based graph search

Public fields

`communities` List of communities

`k` Number of communities

Methods

Public methods:

- `GlobalSearcher$new()`
- `GlobalSearcher$search()`
- `GlobalSearcher$clone()`

Method `new()`**:** Create a new GlobalSearcher

Usage:

```
GlobalSearcher$new(communities, k = 5)
```

Arguments:

`communities` List of Community objects

`k` Number of communities

Method `search()`**:** Search communities

Usage:

```
GlobalSearcher$search(query)
```

Arguments:

`query` Query string

Returns: GlobalSearchResult

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GlobalSearcher$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`GlobalSearchResult` *Global Search Result*

Description

Result from global community search

Public fields

`communities` Matching communities

`summaries` Community summaries

`context` Combined context

`score` Relevance score

Methods

Public methods:

- `GlobalSearchResult$new()`
- `GlobalSearchResult$clone()`

Method `new()`**:** Create new GlobalSearchResult

Usage:

```
GlobalSearchResult$new(
  communities = list(),
  summaries = character(0),
  context = NULL,
  score = 0
)
```

Arguments:

`communities` Communities

`summaries` Summaries

`context` Context

`score` Score

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GlobalSearchResult$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

graphrag *VectrixDB GraphRAG Module*

Description

Native GraphRAG implementation for VectrixDB

Features:

- Entity and relationship extraction
- Hierarchical community detection
- Local, global, and hybrid search strategies
- Incremental graph updates

GraphRAGConfig *GraphRAG Configuration*

Description

Configuration for VectrixDB's GraphRAG implementation

Public fields

`enabled` Whether GraphRAG is enabled

`chunk_size` Target tokens per chunk

`chunk_overlap` Overlapping tokens

`chunk_by_sentence` Preserve sentence boundaries

`extractor` Extraction method

`nlp_model` NLP model name

`llm_provider` LLM provider

`llm_model` Model name

`llm_api_key` API key

`llm_endpoint` Custom endpoint

`llm_temperature` Temperature

`llm_max_tokens` Max tokens

`max_community_levels` Max hierarchy depth

`min_community_size` Min entities per community

`relationship_threshold` Min relationship strength

`deduplicate_entities` Merge similar entities

`entity_similarity_threshold` Similarity for dedup

`search_type` Default search strategy

`local_search_k` Seed entities for local search

`global_search_k` Communities for global search

`traversal_depth` Max hops

`include_relationships` Include relationship context

`include_community_context` Include community summaries

`enable_incremental` Incremental updates

`batch_size` Chunks per batch

`use_cache` Cache embeddings

`cache_ttl` Cache TTL seconds

`entity_types` Types to extract

`relationship_types` Types to extract

Methods

Public methods:

- `GraphRAGConfig$new()`
- `GraphRAGConfig$with_openai()`
- `GraphRAGConfig$with_ollama()`
- `GraphRAGConfig$clone()`

Method `new()`**:** Create a new GraphRAGConfig

Usage:

```
GraphRAGConfig$new(enabled = FALSE, ...)
```

Arguments:

`enabled` Enable GraphRAG

`...` Additional configuration options

Method `with_openai()`**:** Configure for OpenAI

Usage:

```
GraphRAGConfig$with_openai(model = "gpt-4o-mini", api_key = NULL)
```

Arguments:

`model` Model name

`api_key` API key

Returns: Self

Method `with_ollama()`**:** Configure for Ollama

Usage:

```
GraphRAGConfig$with_ollama(
  model = "llama3.2",
  endpoint = "http://localhost:11434"
)
```

Arguments:

`model` Model name

`endpoint` Endpoint URL

Returns: Self

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GraphRAGConfig$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
config <- GraphRAGConfig$new(enabled = TRUE)
db <- Vectrix$new("knowledge_base", graphrag_config = config)

## End(Not run)
```

`GraphRAGPipeline` *GraphRAG Pipeline*

Description

Complete GraphRAG processing pipeline

Public fields

`config` GraphRAGConfig

`graph` KnowledgeGraph

`communities` Detected communities

Methods

Public methods:

- `GraphRAGPipeline$new()`
- `GraphRAGPipeline$process()`
- `GraphRAGPipeline$search()`
- `GraphRAGPipeline$stats()`
- `GraphRAGPipeline$clone()`

Method `new()`**:** Create a new GraphRAGPipeline

Usage:

```
GraphRAGPipeline$new(config = NULL)
```

Arguments:

`config` GraphRAGConfig

Method `process()`**:** Process documents

Usage:

```
GraphRAGPipeline$process(texts, document_ids = NULL)
```

Arguments:

`texts` Character vector of documents

`document_ids` Document IDs

Returns: Self

Method `search()`**:** Search the graph

Usage:

```
GraphRAGPipeline$search(query, search_type = NULL)
```

Arguments:

`query` Query string

`search_type` "local", "global", or "hybrid"

Returns: Search result

Method `stats()`**:** Get statistics

Usage:

```
GraphRAGPipeline$stats()
```

Returns: Named list

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GraphRAGPipeline$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`GraphSearchType` *Graph Search Types*

Description

Graph Search Types

Usage

```
GraphSearchType
```

Format

An object of class `list` of length 3.

hnsw *VectrixDB HNSW Index*

Description

Hierarchical Navigable Small World graph for fast approximate nearest neighbor search

Uses RcppAnnoy for high-performance ANN search. Falls back to brute-force search if RcppAnnoy is not available.

HNSWIndex *HNSW Index*

Description

High-performance approximate nearest neighbor index

Public fields

`dimension` Vector dimension

`metric` Distance metric

`n_trees` Number of trees (for Annoy)

`search_k` Search parameter

Methods

Public methods:

- `HNSWIndex$new()`
- `HNSWIndex$add_items()`
- `HNSWIndex$build()`
- `HNSWIndex$search()`
- `HNSWIndex$get_vector()`
- `HNSWIndex$get_ids()`
- `HNSWIndex$size()`
- `HNSWIndex$remove_items()`
- `HNSWIndex$clear()`
- `HNSWIndex$save()`
- `HNSWIndex$load()`
- `HNSWIndex$clone()`

Method `new()`**:** Create a new HNSWIndex

Usage:

```
HNSWIndex$new(dimension, metric = "angular", n_trees = 50, search_k = -1)
```

Arguments:

`dimension` Vector dimension

`metric` Distance metric: "angular", "euclidean", "manhattan", "dot"

`n_trees` Number of trees for index (higher = more accuracy)

`search_k` Search parameter (higher = more accuracy, -1 = auto)

Method `add_items()`**:** Add items to the index

Usage:

```
HNSWIndex$add_items(ids, vectors)
```

Arguments:

`ids` Character vector of IDs

`vectors` Matrix of vectors (rows = items)

Returns: Self

Method `build()`**:** Build the index (required before searching)

Usage:

```
HNSWIndex$build()
```

Returns: Self

Method `search()`**:** Search for nearest neighbors

Usage:

```
HNSWIndex$search(query, k = 10, include_distances = TRUE)
```

Arguments:

`query` Query vector

`k` Number of neighbors

`include_distances` Return distances

Returns: Data frame with id, distance columns

Method `get_vector()`**:** Get vector by ID

Usage:

```
HNSWIndex$get_vector(id)
```

Arguments:

`id` Item ID

Returns: Vector or NULL

Method `get_ids()`**:** Get all IDs

Usage:

```
HNSWIndex$get_ids()
```

Returns: Character vector

Method `size()`**:** Get item count

Usage:

```
HNSWIndex$size()
```

Returns: Integer

Method `remove_items()`**:** Remove items from index

Usage:

```
HNSWIndex$remove_items(ids)
```

Arguments:

`ids` IDs to remove

Returns: Self

Method `clear()`**:** Clear the index

Usage:

```
HNSWIndex$clear()
```

Returns: Self

Method `save()`**:** Save index to file

Usage:

```
HNSWIndex$save(path)
```

Arguments:

`path` File path

Method `load()`**:** Load index from file

Usage:

```
HNSWIndex$load(path)
```

Arguments:

`path` File path

Returns: Self

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
HNSWIndex$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
# Create index
index <- HNSWIndex$new(dimension = 128, metric = "angular")

# Add vectors
index$add_items(ids = c("a", "b", "c"),
                vectors = matrix(rnorm(384), nrow = 3))
```

```
# Search
results <- index$search(query = rnorm(128), k = 5)

## End(Not run)
```

KeywordAnalyzer *Keyword Analyzer*

Description

Treats entire input as single token

Super class

VectrixDB::TextAnalyzer -> KeywordAnalyzer

Methods

Public methods:

- KeywordAnalyzer$analyze()
- KeywordAnalyzer$clone()

Method `analyze()`**:** Analyze text as single keyword

Usage:

```
KeywordAnalyzer$analyze(text)
```

Arguments:

`text` Input text

Returns: Single-element character vector

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
KeywordAnalyzer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

KnowledgeGraph *Knowledge Graph*

Description

Graph storage for entities and relationships

Public fields

`name` Graph name

Methods

Public methods:

Method `new()`**:** Create a new KnowledgeGraph

Usage:

```
KnowledgeGraph$new(name = "default")
```

Arguments:

`name` Graph name

Method `add_entity()`**:** Add an entity

Usage:

```
KnowledgeGraph$add_entity(entity)
```

Arguments:

`entity` Entity object

Method `add_relationship()`**:** Add a relationship

Usage:

```
KnowledgeGraph$add_relationship(relationship)
```

Arguments:

`relationship` Relationship object

Method `get_entity()`**:** Get entity by ID

Usage:

```
KnowledgeGraph$get_entity(entity_id)
```

Arguments:

`entity_id` Entity ID

Returns: Entity or NULL

Method `get_all_entities()`**:** Get all entities

Usage:

```
KnowledgeGraph$get_all_entities()
```

Returns: List of Entity objects

Method `get_all_relationships()`**:** Get all relationships

Usage:

```
KnowledgeGraph$get_all_relationships()
```

Returns: List of Relationship objects

Method `get_neighbors()`**:** Get neighbors of an entity

Usage:

```
KnowledgeGraph$get_neighbors(entity_id, direction = "both")
```

Arguments:

`entity_id` Entity ID

`direction` "out", "in", or "both"

Returns: List of Entity objects

Method `traverse()`**:** Traverse graph from seed entities

Usage:

```
KnowledgeGraph$traverse(seed_ids, max_depth = 2)
```

Arguments:

`seed_ids` Starting entity IDs

`max_depth` Maximum depth

Returns: SubGraph object

Method `entity_count()`**:** Get entity count

Usage:

```
KnowledgeGraph$entity_count()
```

Returns: Integer

Method `relationship_count()`**:** Get relationship count

Usage:

```
KnowledgeGraph$relationship_count()
```

Returns: Integer

Method `search_entities()`**:** Search entities by name

Usage:

```
KnowledgeGraph$search_entities(query, limit = 10)
```

Arguments:

`query` Query string

`limit` Max results

Returns: List of Entity objects

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
KnowledgeGraph$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`LateInteractionEmbedder` *Late Interaction Embedder (Simplified ColBERT-style)*

Description

Token-level embeddings for late interaction scoring

Public fields

`dimension` Token embedding dimension

`language` Language setting ("en" or "ml")

Methods

Public methods:

- `LateInteractionEmbedder$new()`
- `LateInteractionEmbedder$embed()`
- `LateInteractionEmbedder$score()`
- `LateInteractionEmbedder$clone()`

Method `new()`**:** Create a new LateInteractionEmbedder

Usage:

```
LateInteractionEmbedder$new(dimension = 64, language = "en")
```

Arguments:

`dimension` Embedding dimension per token

`language` Language behavior ("en" = ASCII-focused, "ml" = Unicode-aware)

Method `embed()`**:** Embed texts to token-level embeddings

Usage:

```
LateInteractionEmbedder$embed(texts)
```

Arguments:

`texts` Character vector of texts

Returns: List of matrices (each matrix is token embeddings for a document)

Method `score()`**:** Compute late interaction (MaxSim) score

Usage:

```
LateInteractionEmbedder$score(query_embeddings, doc_embeddings)
```

Arguments:

`query_embeddings` Query token embeddings matrix

`doc_embeddings` Document token embeddings matrix

Returns: Numeric score

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LateInteractionEmbedder$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`LLMProvider` *LLM Provider Types*

Description

LLM Provider Types

Usage

```
LLMProvider
```

Format

An object of class `list` of length 4.

load_hnsw_index *Load HNSW Index*

Description

Load saved index from file

Usage

```
load_hnsw_index(path)
```

Arguments

path	File path

Value

HNSWIndex object

load_word_vectors *Load word vectors into memory*

Description

Loads pre-trained word vectors from a file

Usage

```
load_word_vectors(path, max_words = NULL, normalize = TRUE)
```

Arguments

path	Path to word vectors file (GloVe .txt or word2vec .bin)
max_words	Maximum number of words to load (NULL for all)
normalize	Normalize vectors to unit length

Value

WordVectors object

`LocalSearcher` *Local Searcher*

Description

Entity-based graph search

Public fields

`graph` Knowledge graph

`k` Number of seed entities

`traversal_depth` Max hops

Methods

Public methods:

- `LocalSearcher$new()`
- `LocalSearcher$search()`
- `LocalSearcher$clone()`

Method `new()`**:** Create a new LocalSearcher

Usage:

```
LocalSearcher$new(graph, k = 10, traversal_depth = 2)
```

Arguments:

`graph` KnowledgeGraph

`k` Seed entities

`traversal_depth` Max depth

Method `search()`**:** Search the graph

Usage:

```
LocalSearcher$search(query)
```

Arguments:

`query` Query string

Returns: LocalSearchResult

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LocalSearcher$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`LocalSearchResult` *Local Search Result*

Description

Result from local graph search

Public fields

`entities` Matching entities

`relationships` Related relationships

`subgraph` Traversed subgraph

`context` Combined context text

`score` Relevance score

Methods

Public methods:

- `LocalSearchResult$new()`
- `LocalSearchResult$clone()`

Method `new()`**:** Create new LocalSearchResult

Usage:

```
LocalSearchResult$new(
  entities = list(),
  relationships = list(),
  subgraph = NULL,
  context = NULL,
  score = 0
)
```

Arguments:

`entities` Entities

`relationships` Relationships

`subgraph` SubGraph

`context` Context

`score` Score

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LocalSearchResult$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

MemoryCache *Memory Cache*

Description

In-memory LRU cache with TTL support

Ultra-low latency, limited by available RAM. Best for hot data, session data, frequently accessed vectors.

Super class

VectrixDB::BaseCache -> MemoryCache

Methods

Public methods:

- MemoryCache$new()
- MemoryCache$get()
- MemoryCache$set()
- MemoryCache$delete()
- MemoryCache$exists()
- MemoryCache$clear()
- MemoryCache$size()
- MemoryCache$cleanup_expired()
- MemoryCache$clone()

Method `new()`**:** Create a new MemoryCache

Usage:

```
MemoryCache$new(config = NULL)
```

Arguments:

`config` CacheConfig object

Method `get()`**:** Get value from cache

Usage:

```
MemoryCache$get(key)
```

Arguments:

`key` Cache key

Returns: Value or NULL

Method `set()`**:** Set value in cache

Usage:

```
MemoryCache$set(key, value, ttl = NULL)
```

Arguments:

`key` Cache key

`value` Value to cache

`ttl` Time to live

Method `delete()`**:** Delete key from cache

Usage:

```
MemoryCache$delete(key)
```

Arguments:

`key` Cache key

Returns: Logical success

Method `exists()`**:** Check if key exists

Usage:

```
MemoryCache$exists(key)
```

Arguments:

`key` Cache key

Returns: Logical

Method `clear()`**:** Clear cache

Usage:

```
MemoryCache$clear()
```

Method `size()`**:** Get cache size

Usage:

```
MemoryCache$size()
```

Returns: Integer

Method `cleanup_expired()`**:** Cleanup expired entries

Usage:

```
MemoryCache$cleanup_expired()
```

Returns: Integer count removed

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MemoryCache$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`MMRReranker` *Maximal Marginal Relevance (MMR) Reranker*

Description

Reranks for diversity using MMR algorithm

Public fields

`lambda` Balance between relevance and diversity (0-1)

Methods

Public methods:

- `MMRReranker$new()`
- `MMRReranker$rerank()`
- `MMRReranker$clone()`

Method `new()`**:** Create a new MMRReranker

Usage:

```
MMRReranker$new(lambda = 0.7)
```

Arguments:

`lambda` Relevance vs diversity tradeoff (higher = more relevance)

Method `rerank()`**:** Rerank for diversity

Usage:

```
MMRReranker$rerank(query_vector, doc_vectors, doc_ids, scores, limit = 10)
```

Arguments:

`query_vector` Query embedding
`doc_vectors` Matrix of document embeddings
`doc_ids` Vector of document IDs
`scores` Original relevance scores
`limit` Number of results

Returns: Data frame with reranked results

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MMRReranker$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

NoCache *No-Op Cache*

Description

Disabled cache (no caching)

Super class

VectrixDB::BaseCache -> NoCache

Methods

Public methods:

- NoCache$get()
- NoCache$set()
- NoCache$delete()
- NoCache$exists()
- NoCache$clear()
- NoCache$size()
- NoCache$clone()

Method `get()`**:** Get value from cache (always returns NULL)

Usage:

```
NoCache$get(key)
```

Arguments:

`key` Cache key

Returns: NULL

Method `set()`**:** Set cache value (no-op)

Usage:

```
NoCache$set(key, value, ttl = NULL)
```

Arguments:

`key` Cache key

`value` Value to cache

`ttl` Time-to-live in seconds (ignored)

Returns: Invisibly returns NULL

Method `delete()`**:** Delete key from cache (always FALSE)

Usage:

```
NoCache$delete(key)
```

Arguments:

`key` Cache key

Returns: FALSE

Method `exists()`**:** Check if key exists (always FALSE)

Usage:

```
NoCache$exists(key)
```

Arguments:

`key` Cache key

Returns: FALSE

Method `clear()`**:** Clear cache (no-op)

Usage:

```
NoCache$clear()
```

Returns: Invisibly returns NULL

Method `size()`**:** Get cache size (always 0)

Usage:

```
NoCache$size()
```

Returns: Integer zero

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
NoCache$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`parse_acl` *Parse ACL String*

Description

Parse ACL string like 'user:alice' or 'group:engineering'

Usage

```
parse_acl(acl_string)
```

Arguments

`acl_string`	ACL string

Value

ACLPrincipal object

quick_search *Quick search - Index texts and search immediately*

Description

Quick search - Index texts and search immediately

Usage

```
quick_search(texts, query, limit = 5)
```

Arguments

texts	Character vector of texts to index
query	Search query
limit	Number of results

Value

Results object

Examples

```
## Not run:
results <- quick_search(
  texts = c("Python is great", "Java is verbose", "Rust is fast"),
  query = "programming language"
)
print(results$top()$text)

## End(Not run)
```

RegexExtractor *Regex Entity Extractor*

Description

Simple regex-based entity extractor (no external dependencies)

Public fields

entity_types Entity types to extract

Methods

Public methods:

- RegexExtractor$new()
- RegexExtractor$extract()
- RegexExtractor$clone()

Method `new()`**:** Create a new RegexExtractor

Usage:

```
RegexExtractor$new(entity_types = NULL)
```

Arguments:

`entity_types` Types to extract

Method `extract()`**:** Extract entities from text

Usage:

```
RegexExtractor$extract(text, chunk_id = NULL)
```

Arguments:

`text` Text to extract from

`chunk_id` Chunk ID

Returns: ExtractionResult

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RegexExtractor$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Relationship *Relationship*

Description

A relationship between entities

Public fields

`id` Unique identifier

`source_id` Source entity ID

`target_id` Target entity ID

`type` Relationship type

`description` Description

`weight` Relationship weight

`source_chunks` Source chunk IDs

`metadata` Additional metadata

Methods

Public methods:

- `Relationship$new()`
- `Relationship$to_list()`
- `Relationship$clone()`

Method `new()`**:** Create a new Relationship

Usage:

```
Relationship$new(
  source_id,
  target_id,
  type,
  description = NULL,
  weight = 1,
  source_chunks = NULL,
  metadata = NULL
)
```

Arguments:

`source_id` Source entity

`target_id` Target entity

`type` Relationship type

`description` Description

`weight` Weight

`source_chunks` Sources

`metadata` Metadata

Method `to_list()`**:** Convert to list

Usage:

```
Relationship$to_list()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Relationship$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`reranker` *Advanced Reranking Module*

Description

Learned weight reranking with BM25 + semantic fusion

`RerankerEmbedder` *Reranker (Cross-Encoder Style Scoring)*

Description

Reranks results using term overlap and semantic similarity

Public fields

`language` Language setting ("en" or "ml")

Methods

Public methods:

- `RerankerEmbedder$new()`
- `RerankerEmbedder$score()`
- `RerankerEmbedder$clone()`

Method `new()`**:** Create a new RerankerEmbedder

Usage:

```
RerankerEmbedder$new(language = "en")
```

Arguments:

`language` Language behavior ("en" = English stopwords, "ml" = Unicode tokens)

Method `score()`**:** Score query-document pairs

Usage:

```
RerankerEmbedder$score(query, documents)
```

Arguments:

`query` Query text

`documents` Character vector of document texts

Returns: Numeric vector of scores (0-1)

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RerankerEmbedder$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Result` *Single Search Result*

Description

Represents a single search result with id, text, score, and metadata

Public fields

`id` Document ID

`text` Document text

`score` Relevance score

`metadata` Document metadata

Methods

Public methods:

- `Result$new()`
- `Result$print()`
- `Result$clone()`

Method `new()`**:** Create a new Result object

Usage:

```
Result$new(id, text, score, metadata = list())
```

Arguments:

`id` Document ID

`text` Document text

`score` Relevance score

`metadata` Optional metadata list

Method `print()`**:** Print result summary

Usage:

```
Result$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Result$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Results` *Search Results Collection*

Description

Collection of search results with convenient accessors

Public fields

`items` List of Result objects

`query` Search query

`mode` Search mode

`time_ms` Execution time in ms

Methods

Public methods:

- `Results$new()`
- `Results$length()`
- `Results$texts()`
- `Results$ids()`
- `Results$scores()`
- `Results$top()`
- `Results$get()`
- `Results$foreach()`
- `Results$print()`
- `Results$clone()`

Method `new()`**:** Create a new Results object

Usage:

```
Results$new(items = list(), query = "", mode = "hybrid", time_ms = 0)
```

Arguments:

`items` List of Result objects

`query` Search query string

`mode` Search mode used

`time_ms` Execution time in milliseconds

Method `length()`**:** Get number of results

Usage:

```
Results$length()
```

Method `texts()`**:** Get all result texts

Usage:

```
Results$texts()
```

Returns: Character vector of texts

Method `ids()`**:** Get all result IDs

Usage:

```
Results$ids()
```

Returns: Character vector of IDs

Method `scores()`**:** Get all scores

Usage:

```
Results$scores()
```

Returns: Numeric vector of scores

Method `top()`**:** Get top result

Usage:

```
Results$top()
```

Returns: Result object or NULL if empty

Method `get()`**:** Get result by index

Usage:

```
Results$get(i)
```

Arguments:

`i` Index

Returns: Result object

Method `foreach()`**:** Iterate over results

Usage:

```
Results$foreach(fn)
```

Arguments:

`fn` Function to apply to each result

Method `print()`**:** Print results summary

Usage:

```
Results$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Results$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

SearchMode *Search Mode Enumeration*

Description

Available search modes for VectrixDB

Usage

```
SearchMode
```

Format

An object of class `list` of length 5.

SentenceEmbedder *Sentence Embedder using Word Vectors*

Description

Creates sentence embeddings by averaging word vectors with IDF weighting

Public fields

`dim` Embedding dimension

`vocab_size` Vocabulary size

Methods

Public methods:

- `SentenceEmbedder$new()`
- `SentenceEmbedder$fit()`
- `SentenceEmbedder$embed()`
- `SentenceEmbedder$get_word_vector()`
- `SentenceEmbedder$has_word()`
- `SentenceEmbedder$most_similar()`
- `SentenceEmbedder$clone()`

Method `new()`**:** Create a new SentenceEmbedder

Usage:

```
SentenceEmbedder$new(word_vectors, use_idf = TRUE, smooth_idf = 1)
```

Arguments:

`word_vectors` WordVectors object from load_word_vectors()

`use_idf` Use IDF weighting (recommended)
`smooth_idf` Smoothing for IDF

Method `fit()`**:** Fit IDF weights on a corpus

Usage:

```
SentenceEmbedder$fit(texts)
```

Arguments:

`texts` Character vector of texts

Method `embed()`**:** Embed texts to sentence vectors

Usage:

```
SentenceEmbedder$embed(texts)
```

Arguments:

`texts` Character vector of texts

Returns: Matrix of embeddings (rows are sentences)

Method `get_word_vector()`**:** Get word vector for a single word

Usage:

```
SentenceEmbedder$get_word_vector(word)
```

Arguments:

`word` Word to look up

Returns: Numeric vector or NULL if not found

Method `has_word()`**:** Check if word is in vocabulary

Usage:

```
SentenceEmbedder$has_word(word)
```

Arguments:

`word` Word to check

Returns: Logical

Method `most_similar()`**:** Find most similar words

Usage:

```
SentenceEmbedder$most_similar(word, n = 10)
```

Arguments:

`word` Query word
`n` Number of results

Returns: Data frame with word and similarity

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SentenceEmbedder$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

server *VectrixDB Server Functions*

Description

REST API and dashboard server for VectrixDB

set_cli_config *Set CLI Config*

Description

Set CLI Config

Usage

```
set_cli_config(config)
```

Arguments

config	CLIConfig object

SimpleStemmer *Simple Stemmer*

Description

Simple suffix-stripping stemmer (no external dependencies)

Public fields

`suffixes` List of suffixes to remove

Methods

Public methods:

- `SimpleStemmer$stem()`
- `SimpleStemmer$stem_words()`
- `SimpleStemmer$clone()`

Method `stem()`**:** Stem a word

Usage:

```
SimpleStemmer$stem(word)
```

Arguments:

`word` Word to stem

Returns: Stemmed word

Method `stem_words()`**:** Stem multiple words

Usage:

```
SimpleStemmer$stem_words(words)
```

Arguments:

`words` Character vector

Returns: Stemmed words

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SimpleStemmer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`SparseEmbedder` *Sparse Embedder (BM25/TF-IDF)*

Description

Generates sparse BM25 embeddings for keyword search

Public fields

`vocab` Vocabulary

`language` Language setting ("en" or "ml")

Methods

Public methods:

- `SparseEmbedder$new()`
- `SparseEmbedder$fit()`
- `SparseEmbedder$embed()`
- `SparseEmbedder$query_terms()`
- `SparseEmbedder$clone()`

Method `new()`**:** Create a new SparseEmbedder

Usage:

```
SparseEmbedder$new(language = "en")
```

Arguments:

`language` Language behavior ("en" = ASCII-focused, "ml" = Unicode-aware)

Method `fit()`**:** Fit the embedder on a corpus

Usage:

```
SparseEmbedder$fit(texts)
```

Arguments:

`texts` Character vector of texts

Method `embed()`**:** Embed texts to sparse vectors

Usage:

```
SparseEmbedder$embed(texts)
```

Arguments:

`texts` Character vector of texts

Returns: Sparse matrix of BM25 scores

Method `query_terms()`**:** Get term scores for a query

Usage:

```
SparseEmbedder$query_terms(query)
```

Arguments:

`query` Query text

Returns: Named vector of term scores

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SparseEmbedder$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`storage` *VectrixDB Storage Classes*

Description

Storage backends for VectrixDB

SubGraph *SubGraph*

Description

A subset of a knowledge graph

Public fields

`entities` Entities in subgraph

`relationships` Relationships in subgraph

Methods

Public methods:

- `SubGraph$new()`
- `SubGraph$to_list()`
- `SubGraph$clone()`

Method `new()`**:** Create a new SubGraph

Usage:

```
SubGraph$new(entities = list(), relationships = list())
```

Arguments:

`entities` Entities

`relationships` Relationships

Method `to_list()`**:** Convert to list

Usage:

```
SubGraph$to_list()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SubGraph$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

text_analyzer_english *Create English Text Analyzer*

Description

English analyzer with stemming and stopwords

Usage

```
text_analyzer_english()
```

Value

TextAnalyzer object

text_analyzer_keyword *Create Keyword Text Analyzer*

Description

No tokenization - treat input as single token

Usage

```
text_analyzer_keyword()
```

Value

TextAnalyzer object

text_analyzer_simple *Create Simple Text Analyzer*

Description

Lowercase + letter-only tokenization

Usage

```
text_analyzer_simple()
```

Value

TextAnalyzer object

text_analyzer_standard *Create Standard Text Analyzer*

Description

Lowercase + basic tokenization

Usage

```
text_analyzer_standard()
```

Value

TextAnalyzer object

TextAnalyzer *Text Analyzer*

Description

Text analyzer for search indexing

Provides text processing pipelines:

- Tokenization
- Lowercasing
- Stopword removal
- Stemming
- Synonym expansion

Public fields

`lowercase` Convert to lowercase

`remove_stopwords` Remove stopwords

`stopwords` Set of stopwords

`stemmer` Stemmer object

`synonyms` Synonym dictionary

`min_token_length` Minimum token length

`max_token_length` Maximum token length

`token_pattern` Regex pattern for tokens

Methods

Public methods:

- TextAnalyzer$new()
- TextAnalyzer$analyze()
- TextAnalyzer$analyze_query()
- TextAnalyzer$clone()

Method `new()`**:** Create a new TextAnalyzer

Usage:

```
TextAnalyzer$new(
  lowercase = TRUE,
  remove_stopwords = FALSE,
  stopwords = NULL,
  use_stemmer = FALSE,
  synonyms = NULL,
  min_token_length = 1,
  max_token_length = 100,
  token_pattern = "[a-zA-Z0-9]+"
)
```

Arguments:

`lowercase` Lowercase text (default: TRUE)

`remove_stopwords` Remove stopwords (default: FALSE)

`stopwords` Custom stopwords (default: ENGLISH_STOPWORDS)

`use_stemmer` Use stemming (default: FALSE)

`synonyms` Named list of synonyms

`min_token_length` Min length (default: 1)

`max_token_length` Max length (default: 100)

`token_pattern` Regex pattern

Method `analyze()`**:** Analyze text and return tokens

Usage:

```
TextAnalyzer$analyze(text)
```

Arguments:

`text` Input text

Returns: Character vector of tokens

Method `analyze_query()`**:** Analyze a query string

Usage:

```
TextAnalyzer$analyze_query(query)
```

Arguments:

`query` Query text

Returns: Character vector of tokens

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TextAnalyzer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
analyzer <- TextAnalyzer$english()
tokens <- analyzer$analyze("The quick brown foxes are jumping")
# c("quick", "brown", "fox", "jump")

## End(Not run)
```

`TextUnit` *Text Unit*

Description

A chunk of text from a document

Public fields

`id` Unique identifier

`text` Text content

`document_id` Source document

`chunk_index` Index in document

`start_char` Start position

`end_char` End position

`metadata` Additional metadata

Methods

Public methods:

- `TextUnit$new()`
- `TextUnit$clone()`

Method `new()`**:** Create a new TextUnit

Usage:

```
TextUnit$new(
  id,
  text,
  document_id = NULL,
  chunk_index = 0,
  start_char = 0,
  end_char = 0,
  metadata = NULL
)
```

Arguments:

`id` Unique ID

`text` Content

`document_id` Source doc

`chunk_index` Index

`start_char` Start

`end_char` End

`metadata` Metadata

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TextUnit$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`vdb_add` *Add Documents*

Description

Add documents to a collection

Usage

```
vdb_add(db, texts, metadata = NULL, ids = NULL)
```

Arguments

`db`	Vectrix object or collection name
`texts`	Character vector of texts
`metadata`	Optional metadata
`ids`	Optional IDs

Value

Vectrix object

Examples

```
## Not run:
vdb_add(db, c("Document 1", "Document 2"))
vdb_add("my_docs", c("Another doc"))

## End(Not run)
```

---

vdb_add_dir *Batch Add from Directory*

Description

Add all text files from a directory

Usage

```
vdb_add_dir(db, dir_path, pattern = "\\.txt$", recursive = TRUE)
```

Arguments

db	Vectrix object or collection name
dir_path	Directory path
pattern	File pattern (default: "*.txt")
recursive	Search subdirectories

Value

Vectrix object

Examples

```
## Not run:
vdb_add_dir(db, "./documents/")

## End(Not run)
```

vdb_create *Create Collection*

Description

Create a new VectrixDB collection

Usage

```
vdb_create(name, model = "tfidf", dimension = NULL, data_dir = NULL)
```

Arguments

name	Collection name
model	Embedding model
dimension	Vector dimension
data_dir	Data directory

Value

Vectrix object

Examples

```
## Not run:
db <- vdb_create("my_docs")

## End(Not run)
```

vdb_dashboard *Launch VectrixDB Dashboard*

Description

Start the VectrixDB API server and mount the HTML dashboard at `/dashboard`.

Usage

```
vdb_dashboard(
  db = NULL,
  data_path = NULL,
  port = 7377,
  host = "127.0.0.1",
  launch.browser = TRUE,
  api_key = NULL
)
```

Arguments

db	Optional Vectrix object.
data_path	Path to vector database directory.
port	Port number (default: 7377).
host	Host address (default: "127.0.0.1").
launch.browser	Whether to open browser on start.
api_key	Optional API key for authenticated write operations.

Value

Invisibly returns server object from vectrix_serve().

---

vdb_dashboard_simple *Launch Simple Dashboard*

Description

Convenience wrapper around vdb_dashboard() using db$path.

Usage

```
vdb_dashboard_simple(db)
```

Arguments

db	Vectrix object.

Value

Invisibly returns server object from vdb_dashboard().

---

vdb_delete *Delete Collection*

Description

Delete a collection

Usage

```
vdb_delete(name, data_dir = NULL, confirm = TRUE)
```

Arguments

name	Collection name
data_dir	Data directory
confirm	Require confirmation

Value

Logical success

Examples

```
## Not run:
vdb_delete("my_docs")

## End(Not run)
```

---

vdb_delete_docs *Delete Documents*

Description

Delete documents by ID

Usage

```
vdb_delete_docs(db, ids)
```

Arguments

db	Vectrix object or collection name
ids	Document ID(s)

Value

Vectrix object

vdb_export *Export Collection*

Description

Export collection to JSON file

Usage

```
vdb_export(db, path)
```

Arguments

db	Vectrix object or collection name
path	Output file path

Value

Logical success

Examples

```
## Not run:
vdb_export(db, "backup.json")

## End(Not run)
```

vdb_get *Get Document*

Description

Get document by ID

Usage

```
vdb_get(db, ids)
```

Arguments

db	Vectrix object or collection name
ids	Document ID(s)

Value

List of Result objects

vdb_import *Import from File*

Description

Import documents from text file

Usage

```
vdb_import(db, path, separator = "\n")
```

Arguments

db	Vectrix object or collection name
path	Input file path
separator	Line separator for documents

Value

Vectrix object

Examples

```
## Not run:
vdb_import(db, "documents.txt")

## End(Not run)
```

vdb_info *Collection Info*

Description

Get collection information

Usage

```
vdb_info(db)
```

Arguments

db	Vectrix object or collection name

Value

Named list of info

Examples

```
## Not run:
vdb_info(db)
vdb_info("my_docs")

## End(Not run)
```

`vdb_interactive` *Start Interactive CLI*

Description

Start an interactive VectrixDB session

Usage

```
vdb_interactive(collection = NULL)
```

Arguments

`collection`	Default collection name

Examples

```
## Not run:
vdb_interactive()

## End(Not run)
```

`vdb_list` *List Collections*

Description

List all VectrixDB collections in the data directory

Usage

```
vdb_list(data_dir = NULL)
```

Arguments

`data_dir`	Data directory path

Value

Character vector of collection names

Examples

```
## Not run:
vdb_list()

## End(Not run)
```

---

vdb_open *Open Collection*

Description

Open an existing collection

Usage

```
vdb_open(name, data_dir = NULL)
```

Arguments

name	Collection name
data_dir	Data directory

Value

Vectrix object

Examples

```
## Not run:
db <- vdb_open("my_docs")

## End(Not run)
```

---

vdb_search *Search Collection*

Description

Search a collection

Usage

```
vdb_search(db, query, limit = 10, mode = "hybrid", show = TRUE)
```

Arguments

db	Vectrix object or collection name
query	Search query
limit	Number of results
mode	Search mode: "dense", "sparse", "hybrid", "ultimate"
show	Print results

Value

Results object

Examples

```
## Not run:
results <- vdb_search(db, "machine learning")
results <- vdb_search("my_docs", "AI", limit = 5)

## End(Not run)
```

---

vdb_stats *Collection Statistics*

Description

Get detailed statistics

Usage

```
vdb_stats(db)
```

Arguments

db	Vectrix object or collection name

Value

Named list of stats

VectorCache *Vector Cache*

Description

Specialized cache for vector search results

Features:

- Query result caching
- Vector embedding caching
- Automatic cache invalidation

Public fields

`prefix` Cache key prefix

Methods

Public methods:

- `VectorCache$new()`
- `VectorCache$get_search_results()`
- `VectorCache$set_search_results()`
- `VectorCache$get_vector()`
- `VectorCache$set_vector()`
- `VectorCache$invalidate_vector()`
- `VectorCache$stats()`
- `VectorCache$clone()`

Method `new()`**:** Create a new VectorCache

Usage:

```
VectorCache$new(cache, prefix = "vec:")
```

Arguments:

`cache` Base cache backend

`prefix` Key prefix (default: "vec:")

Method `get_search_results()`**:** Get cached search results

Usage:

```
VectorCache$get_search_results(collection, query, filter = NULL, limit = 10)
```

Arguments:

`collection` Collection name

`query` Query vector

`filter` Filter conditions

`limit` Result limit

Returns: Cached results or NULL

Method `set_search_results()`**:** Cache search results

Usage:

```
VectorCache$set_search_results(
  collection,
  query,
  results,
  filter = NULL,
  limit = 10,
  ttl = 300
)
```

Arguments:

`collection` Collection name

`query` Query vector

`results` Search results

`filter` Filter conditions

`limit` Result limit

`ttl` Time to live (default: 300)

Method `get_vector()`**:** Get cached vector

Usage:

```
VectorCache$get_vector(collection, vector_id)
```

Arguments:

`collection` Collection name

`vector_id` Vector ID

Returns: Cached vector data or NULL

Method `set_vector()`**:** Cache vector data

Usage:

```
VectorCache$set_vector(collection, vector_id, data, ttl = 3600)
```

Arguments:

`collection` Collection name

`vector_id` Vector ID

`data` Vector data

`ttl` Time to live (default: 3600)

Method `invalidate_vector()`**:** Invalidate cached vector

Usage:

```
VectorCache$invalidate_vector(collection, vector_id)
```

Arguments:

`collection` Collection name

`vector_id` Vector ID

Method `stats()`**:** Get cache statistics

Usage:

```
VectorCache$stats()
```

Returns: CacheStats object

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
VectorCache$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Vectrix` *VectrixDB Easy API - The Simplest Vector Database*

Description

Zero config. Text in, results out. One line for everything.

Public fields

`name` Collection name

`path` Storage path

`dimension` Vector dimension

`model_name` Model identifier

`model_type` Model type

`language` Language setting

`tier` Storage tier

Methods

Public methods:

- `Vectrix$new()`
- `Vectrix$add()`
- `Vectrix$set_language()`
- `Vectrix$search()`
- `Vectrix$delete()`
- `Vectrix$clear()`
- `Vectrix$count()`
- `Vectrix$get()`
- `Vectrix$similar()`
- `Vectrix$close()`
- `Vectrix$print()`

- Vectrix$clone()

Method `new()`**:** Create or open a VectrixDB collection

Usage:

```
Vectrix$new(
  name = "default",
  path = NULL,
  model = NULL,
  dimension = NULL,
  embed_fn = NULL,
  model_path = NULL,
  language = NULL,
  tier = "dense",
  auto_download = TRUE
)
```

Arguments:

`name` Collection name

`path` Storage path. Defaults to a session temp directory.

`model` Embedding model: "tfidf" (default), "glove-50", "glove-100", "glove-200", "glove-300", or "word2vec"

`dimension` Vector dimension (auto-detected for GloVe)

`embed_fn` Custom embedding function: fn(texts) -> matrix

`model_path` Path to pre-trained word vectors (GloVe .txt or word2vec .bin)

`language` Language behavior: "en" (English-focused) or "ml" (multilingual/Unicode)

`tier` Storage tier: "dense", "hybrid", "ultimate", or "graph"

`auto_download` Automatically download GloVe vectors if needed (default: TRUE)

Examples:

```
\dontrun{
# Default TF-IDF embeddings (no external files needed)
db <- Vectrix$new("docs")

# With GloVe 100d word vectors (auto-downloads ~130MB)
db <- Vectrix$new("docs", model = "glove-100")

# With pre-downloaded GloVe
db <- Vectrix$new("docs", model_path = "path/to/glove.6B.100d.txt")

# Custom embedding function
db <- Vectrix$new("docs", embed_fn = my_embed_function, dimension = 768)
}
```

Method `add()`**:** Add texts to the collection

Usage:

```
Vectrix$add(texts, metadata = NULL, ids = NULL)
```

Arguments:

`texts` Single text or character vector of texts

`metadata` Optional metadata list or list of lists

`ids` Optional custom IDs

Returns: Self for chaining

Examples:

```
\dontrun{
db$add(c("text 1", "text 2"))
db$add("another text", metadata = list(source = "web"))
}
```

Method `set_language()`**:** Update collection language behavior

Usage:

```
Vectrix$set_language(language = "en")
```

Arguments:

`language` Language behavior: "en" or "ml"

Returns: Self for chaining

Method `search()`**:** Search the collection

Usage:

```
Vectrix$search(
  query,
  limit = 10,
  mode = "hybrid",
  rerank = NULL,
  filter = NULL,
  diversity = 0.7
)
```

Arguments:

`query` Search query text

`limit` Number of results (default: 10)

`mode` Search mode: "dense", "sparse", "hybrid", "ultimate"

`rerank` Reranking method: NULL, "mmr", "exact", "cross-encoder"

`filter` Metadata filter

`diversity` Diversity parameter for MMR (0-1)

Returns: Results object with search results

Examples:

```
\dontrun{
results <- db$search("python programming")
results <- db$search("AI", mode = "ultimate", rerank = "mmr")
print(results$top()$text)
}
```

Method `delete()`**:** Delete documents by ID

Usage:

```
Vectrix$delete(ids)
```

Arguments:

`ids` Document ID(s) to delete

Returns: Self for chaining

Method `clear()`**:** Clear all documents from collection

Usage:

```
Vectrix$clear()
```

Returns: Self for chaining

Method `count()`**:** Get number of documents

Usage:

```
Vectrix$count()
```

Returns: Integer count

Method `get()`**:** Get documents by ID

Usage:

```
Vectrix$get(ids)
```

Arguments:

`ids` Document ID(s)

Returns: List of Result objects

Method `similar()`**:** Find similar documents to a given document

Usage:

```
Vectrix$similar(id, limit = 10)
```

Arguments:

`id` Document ID

`limit` Number of results

Returns: Results object

Method `close()`**:** Close the database connection

Usage:

```
Vectrix$close()
```

Method `print()`**:** Print Vectrix summary

Usage:

```
Vectrix$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Vectrix$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
# Create and add - ONE LINE
db <- Vectrix$new("my_docs")$add(c("Python is great", "Machine learning is fun"))

# Search - ONE LINE
results <- db$search("programming")

# Full power - STILL ONE LINE
results <- db$search("AI", mode = "ultimate")  # dense + sparse + rerank

## End(Not run)


## ------------------------------------------------
## Method `Vectrix$new`
## ------------------------------------------------

## Not run:
# Default TF-IDF embeddings (no external files needed)
db <- Vectrix$new("docs")

# With GloVe 100d word vectors (auto-downloads ~130MB)
db <- Vectrix$new("docs", model = "glove-100")

# With pre-downloaded GloVe
db <- Vectrix$new("docs", model_path = "path/to/glove.6B.100d.txt")

# Custom embedding function
db <- Vectrix$new("docs", embed_fn = my_embed_function, dimension = 768)

## End(Not run)

## ------------------------------------------------
## Method `Vectrix$add`
## ------------------------------------------------

## Not run:
db$add(c("text 1", "text 2"))
db$add("another text", metadata = list(source = "web"))

## End(Not run)

## ------------------------------------------------
## Method `Vectrix$search`
## ------------------------------------------------

## Not run:
results <- db$search("python programming")
results <- db$search("AI", mode = "ultimate", rerank = "mmr")
print(results$top()$text)
```

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

`vectrix_create` *Create a new Vectrix collection*

Description

Create a new Vectrix collection

Usage

```
vectrix_create(name = "default", ...)
```

Arguments

`name`	Collection name
`...`	Additional arguments passed to Vectrix$new()

Value

Vectrix object

`vectrix_info` *Display VectrixDB information*

Description

Show database statistics and info

Usage

```
vectrix_info(path = NULL)
```

Arguments

`path`	Database path

Examples

```
## Not run:
vectrix_info(file.path(tempdir(), "my_data"))

## End(Not run)
```

vectrix_open *Open an existing Vectrix collection*

Description

Open an existing Vectrix collection

Usage

```
vectrix_open(name = "default", path = NULL)
```

Arguments

name	Collection name
path	Storage path

Value

Vectrix object

vectrix_serve *Start VectrixDB server*

Description

Launch a REST API server with optional dashboard

Usage

```
vectrix_serve(
  path = NULL,
  host = "127.0.0.1",
  port = 7377,
  api_key = NULL,
  dashboard = TRUE,
  launch.browser = FALSE
)
```

Arguments

path	Database path
host	Host address (default: "127.0.0.1")
port	Port number (default: 7377)
api_key	Optional API key for authentication
dashboard	Enable dashboard (default: TRUE)
launch.browser	Open dashboard/docs URL in browser (default: FALSE)

Value

Invisible NULL (server runs until stopped)

Examples

```
## Not run:
vectrix_serve(path = file.path(tempdir(), "my_data"), port = 7377)

## End(Not run)
```

VectrixDB *VectrixDB Database Class*

Description

Main database interface managing collections

Usage

```
vectrixdb(path = NULL, storage_type = "memory")
```

Arguments

path	Storage path
storage_type	Storage type

Value

VectrixDB object

Public fields

`path` Database storage path

Methods

Public methods:

- VectrixDB$new()
- VectrixDB$create_collection()
- VectrixDB$get_collection()
- VectrixDB$list_collections()
- VectrixDB$delete_collection()
- VectrixDB$has_collection()
- VectrixDB$stats()
- VectrixDB$close()
- VectrixDB$print()

- VectrixDB$clone()

Method `new()`**:** Create or open a VectrixDB database

Usage:

```
VectrixDB$new(path = NULL, storage_type = "memory")
```

Arguments:

`path` Storage path

`storage_type` Storage type ("memory" or "sqlite")

Method `create_collection()`**:** Create a new collection

Usage:

```
VectrixDB$create_collection(
  name,
  dimension,
  metric = "cosine",
  enable_text_index = TRUE
)
```

Arguments:

`name` Collection name

`dimension` Vector dimension

`metric` Distance metric

`enable_text_index` Enable text indexing

Returns: Collection object

Method `get_collection()`**:** Get an existing collection

Usage:

```
VectrixDB$get_collection(name)
```

Arguments:

`name` Collection name

Returns: Collection object

Method `list_collections()`**:** List all collections

Usage:

```
VectrixDB$list_collections()
```

Returns: Character vector of collection names

Method `delete_collection()`**:** Delete a collection

Usage:

```
VectrixDB$delete_collection(name)
```

Arguments:

`name` Collection name

Method `has_collection()`**:** Check if collection exists

Usage:

```
VectrixDB$has_collection(name)
```

Arguments:

`name` Collection name

Returns: Logical

Method `stats()`**:** Get database statistics

Usage:

```
VectrixDB$stats()
```

Returns: List with stats

Method `close()`**:** Close the database

Usage:

```
VectrixDB$close()
```

Method `print()`**:** Print database summary

Usage:

```
VectrixDB$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
VectrixDB$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`word_vectors` *Word Vector Management*

Description

Download, load, and use pre-trained word vectors (GloVe, fastText)

Index