

Package: fastEmbedR (via r-universe)

September 30, 2026

Type Package

Title Native UMAP and Interpolation-Based t-SNE Embeddings

Version 0.99.17

Copyright See inst/COPYRIGHTS and inst/NOTICE for source-level ownership, adapted components, and optional linked dependencies.

Description A native float32 UMAP implementation and a configurable interpolation-based t-SNE implementation for embeddings from data matrices or precomputed nearest-neighbor indices and distances. One-call workflows use package-native CPU HNSW and Apple Metal exact or IVF-Flat search. Optional CUDA support links directly to FAISS GPU and the RAPIDS cuVS C API while retaining nearest-neighbor results on the device. Preprocessing, graph or affinity construction, fixed-reference transform, and layout optimization use native code with explicit CPU, Metal, and CUDA backend reporting. UMAP exposes neighborhood, metric, graph mode, backend, seed, and preprocessing controls, while optimizer controls such as epochs, minimum distance, spread, learning rate, repulsion, and negative sampling follow a recorded release policy rather than a general-purpose sweep API. Optional graph utilities support downstream Louvain, Leiden, and Walktrap community detection.

License MIT + file LICENSE

URL <https://github.com/tkcaccia/fastEmbedR>,
<https://tkcaccia.github.io/fastEmbedR/>,
<https://github.com/tkcaccia/fastEmbedR-benchmark>

BugReports <https://github.com/tkcaccia/fastEmbedR/issues>

biocViews Software, Infrastructure, Preprocessing, DimensionReduction, Visualization, Clustering, SingleCell, FlowCytometry, GeneExpression, GPU

Encoding UTF-8

Imports Biobase, methods, Rcpp, withr

Suggests float, igraph, jsonlite, knitr, RhpCBLASctl, rmarkdown, testthat

LinkingTo Rcpp

SystemRequirements C++17; optional Metal backend: Apple Silicon with macOS 14 or later and Xcode 15 or later; the required Foundation, Accelerate, Metal, Metal Performance Shaders, and Metal Performance Shaders Graph frameworks are supplied by the Apple SDK; optional CUDA backend: CUDA Toolkit with cuFFT, cuBLAS, cuSOLVER, and CUB/Thrust (CCCL); FAISS GPU and RAPIDS cuVS for CUDA nearest-neighbor search; RAPIDS RAFT and RMM only when CUDA TSVD initialization is enabled.

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Config/pak/sysreqs nvidia-cuda-dev libfaiss-dev

Repository <https://biocstaging.r-universe.dev>

Date/Publication 2026-08-30 12:59:37 UTC

RemoteUrl <https://github.com/BiocStaging/fastEmbedR>

RemoteRef HEAD

RemoteSha 918492c82e4804299b8713b5315b0fcd6f47d2

Contents

embed_knn	3
evaluate_embedding	4
fastEmbedR_api	6
fastEmbedR_backend	6
fastEmbedR_capabilities	7
fastEmbedR_embedding_methods	8
fastEmbedR_graph_methods	9
fit_landmark_model	9
graph_cluster	11
knn_graph	12
landmark_tsne	13
landmark_umap	15
pca	17
precompute_knn	18
precompute_query_knn	20
prepare_tsne_knn	21
prepare_umap_knn	22
print.fastEmbedR_knn	23
print.fastEmbedR_umap_initialization	24
project_landmark_model	24

<i>embed_knn</i>	3
------------------	---

select_landmarks	26
transform_tsne	26
tsne	29
tsne_knn	32
tsne_pca_init	34
umap	35
umap_init	37
umap_knn	39

Index	41
--------------	-----------

embed_knn	<i>Embed from precomputed KNN</i>
-----------	-----------------------------------

Description

Embed from precomputed KNN

Usage

```
embed_knn(  
  indices,  
  distances = NULL,  
  method = "umap",  
  n_components = 2L,  
  seed = 4L,  
  verbose = FALSE,  
  backend = NULL,  
  n.cores = NULL,  
  ...  
)
```

Arguments

indices	A list containing KNN indices and distances, or an integer KNN index matrix. If a self-neighbor first column is present it is removed automatically.
distances	Numeric KNN distance matrix matching indices. Leave as NULL when indices is a KNN list.
method	Embedding method: "tsne" or "umap".
n_components	Output dimensionality.
seed	Random seed.
verbose	Print native optimizer progress.
backend	Execution backend: "cpu", "cuda", or "metal".
n.cores	Number of CPU cores used by the CPU optimizer. Native GPU optimizers ignore this argument.
...	Additional t-SNE-specific parameters such as perplexity, iteration controls, Y_init, learning-rate controls, and exaggeration controls.

Value

A numeric embedding matrix with resolved settings stored in `attr(layout, "fastEmbedR_config")`.

Examples

```
x <- scale(as.matrix(iris[1:30, 1:4]))
d <- as.matrix(stats::dist(x))
diag(d) <- Inf
k <- 5L
indices <- t(apply(d, 1L, order))[, seq_len(k), drop = FALSE]
distances <- matrix(
  d[cbind(rep(seq_len(nrow(d)), each = k), as.vector(t(indices)))],
  nrow = nrow(d), byrow = TRUE
)
layout <- embed_knn(indices, distances, method = "umap", seed = 1)
```

evaluate_embedding	<i>Evaluate an embedding against high-dimensional structure</i>
--------------------	---

Description

Trustworthiness and continuity use their standard rank-penalty definitions. Local rank metrics are computed exactly on a deterministic subsample so the function does not allocate all-pairs matrices for the complete dataset. Float32 inputs are sampled before conversion to a dense double matrix.

Usage

```
evaluate_embedding(
  x_high,
  embedding,
  labels = NULL,
  batch = NULL,
  k = c(15L, 30L, 50L),
  primary_k = NULL,
  reference_nn = NULL,
  sample_size_for_global_metrics = min(5000L, nrow(x_high)),
  sample_size_for_local_metrics = min(2000L, nrow(x_high)),
  use_cache = FALSE,
  cache_dir = file.path("results", "cache"),
  force_recompute = FALSE,
  seed = NA_integer_,
  method = NA_character_,
  backend = NA_character_,
  n.cores = NULL,
  dataset = NA_character_
)
```

Arguments

<code>x_high</code>	High-dimensional input, usually scaled data or PCA scores.
<code>embedding</code>	Low-dimensional embedding matrix.
<code>labels</code>	Optional biological/class labels.
<code>batch</code>	Optional batch labels.
<code>k</code>	Neighbor sizes used for structure metrics.
<code>primary_k</code>	Neighbor size used for scalar trustworthiness, continuity, label accuracy, rare-class recall, density summaries, and rank-error metrics. The default NULL preserves the historical <code>k = 15</code> behavior when possible.
<code>reference_nn</code>	Optional precomputed high-dimensional nearest-neighbor list. It may include a self-neighbor column.
<code>sample_size_for_global_metrics</code>	Maximum deterministic subsample size for all-pairs global distance metrics.
<code>sample_size_for_local_metrics</code>	Maximum deterministic subsample size for local trustworthiness, continuity, and neighbor preservation metrics.
<code>use_cache</code>	Cache the high-dimensional reference neighbors used by quality metrics.
<code>cache_dir</code>	Directory for quality-metric reference-neighbor cache files.
<code>force_recompute</code>	Ignore cached quality reference neighbors.
<code>seed</code>	Random seed recorded in the output and used for subsampling.
<code>method</code>	Method name recorded in the output.
<code>backend</code>	Backend name recorded in the output.
<code>n.cores</code>	Number of CPU cores used when quality metrics need nearest-neighbor searches. Ignored by native GPU metric backends.
<code>dataset</code>	Dataset name recorded in the output.

Value

A one-row data frame with local, global, label-aware, batch-aware, and metadata columns. Per-class recall is also attached as an attribute.

Examples

```
x <- scale(as.matrix(iris[, 1:4]))
metrics <- evaluate_embedding(
  x, x[, 1:2],
  labels = iris$Species, k = 5,
  sample_size_for_local_metrics = 100,
  sample_size_for_global_metrics = 100,
  seed = 1
)
```

fastEmbedR_api	<i>Inspect the supported public API</i>
----------------	---

Description

fastEmbedR_api() returns the package's machine-readable public API map. The map distinguishes canonical workflows from advanced reusable-state interfaces, diagnostics, secondary graph utilities, and compatibility helpers. It also records accepted input classes, returned classes, backend support, device-residency behavior, related S3 methods, and lifecycle status.

Usage

```
fastEmbedR_api()
```

Details

The inventory describes the public contract of the installed package. A backend marked as supported still requires that backend to have been compiled and to be available at runtime; use [fastEmbedR_capabilities\(\)](#) to inspect the current installation.

UMAP rows describe fastEmbedR's fixed optimizer policy, not a general-purpose parameter-sweep interface. See [umap\(\)](#) for the public/internal control boundary and the resolved values recorded by each fit.

Value

A data frame with one row per exported function.

Examples

```
api <- fastEmbedR_api()
api[api$tier == "canonical", c("function", "purpose", "return_class")]
```

fastEmbedR_backend	<i>Configure the default fastEmbedR execution backend</i>
--------------------	---

Description

Configure the default fastEmbedR execution backend

Usage

```
fastEmbedR_backend(backend = NULL)
```

Arguments

backend Optional backend: "cpu", "cuda", or "metal".

Value

The active backend. Setting returns the previous option invisibly.

Examples

```
old_backend <- getOption("fastEmbedR.backend")
fastEmbedR_backend("cpu")
fastEmbedR_backend()
options(fastEmbedR.backend = old_backend)
```

`fastEmbedR_capabilities`*Report compiled fastEmbedR backend capabilities*

Description

Reports whether the installed fastEmbedR build can provide nearest-neighbor search, embedding optimization, and graph clustering through each native execution backend. This function performs capability checks only; it does not change the backend selected by `fastEmbedR_backend()`.

Usage

```
fastEmbedR_capabilities()
```

Details

The `cuvs` row describes the RAPIDS cuVS nearest-neighbor component used by the CUDA backend. It is diagnostic information and is not itself a value accepted by public backend arguments.

Value

A data frame with one row per backend or backend component. In addition to logical availability columns, the result records the public backend name, detected device, native KNN engine, internal precision, linked runtime families, and an explanation when a component is unavailable. Device-name queries are best effort and do not change the capability decision made by the compiled native probes.

Examples

```
fastEmbedR_capabilities()
```

fastEmbedR_embedding_methods

Print or plot an embedding result

Description

These methods summarize a fastEmbedR_embedding object or plot its first two layout dimensions.

Usage

```
## S3 method for class 'fastEmbedR_embedding'
print(x, ...)

## S3 method for class 'fastEmbedR_embedding'
plot(
  x,
  labels = x$labels,
  pch = 21,
  bg = NULL,
  col = "black",
  xlab = "Component 1",
  ylab = "Component 2",
  main = NULL,
  ...
)
```

Arguments

x	A fastEmbedR_embedding object.
...	Additional arguments passed to <code>graphics::plot()</code> by the plot method. The print method ignores them.
labels	Optional labels used to color plotted observations.
pch	Plotting symbol passed to <code>graphics::plot()</code> .
bg	Point background colors. When NULL, integer colors are derived from labels, or white is used when labels are absent.
col	Point border or foreground color.
xlab, ylab	Axis labels.
main	Plot title. NULL derives a title from the fitted method.

Value

The input object, invisibly.

fastEmbedR_graph_methods

Print graph and graph-clustering results

Description

Print graph and graph-clustering results

Usage

```
## S3 method for class 'fastEmbedR_graph'
print(x, ...)
```

```
## S3 method for class 'fastEmbedR_graph_cluster'
print(x, ...)
```

Arguments

x	A fastEmbedR_graph or fastEmbedR_graph_cluster object.
...	Unused.

Value

x, invisibly.

fit_landmark_model

Fit an embedding model on selected landmarks

Description

fit_landmark_model() runs the ordinary production [umap\(\)](#) or [tsne\(\)](#) implementation on the selected reference rows. The returned object stores the reference data and embedding needed to project the complementary rows or genuinely new observations.

Usage

```
fit_landmark_model(
  data,
  selection,
  method = c("umap", "tsne"),
  n_neighbors = NULL,
  perplexity = NULL,
  affinity_support = c("standard", "compact"),
  n_components = 2L,
  metric = c("euclidean", "cosine", "correlation", "inner_product"),
```

```

    seed = 4L,
    backend = NULL,
    n.cores = NULL,
    graph_mode = c("fuzzy", "binary"),
    keep_knn = FALSE,
    verbose = FALSE,
    ...
  )

```

Arguments

<code>data</code>	Data in the analysis space used for KNN. Preprocess once before calling this staged API when centering, scaling, or PCA is required.
<code>selection</code>	A result from <code>select_landmarks()</code> or explicit landmark row indices.
<code>method</code>	"umap" or "tsne".
<code>n_neighbors</code>	UMAP neighborhood size. For t-SNE this is the precomputed KNN support width; NULL derives it from perplexity.
<code>perplexity</code>	t-SNE perplexity.
<code>affinity_support</code>	t-SNE affinity support policy. "standard" uses <code>ceiling(3 * perplexity)</code> neighbors; "compact" is the explicit <code>ceiling(perplexity)</code> approximation.
<code>n_components</code>	Embedding dimensionality.
<code>metric</code>	KNN metric.
<code>seed</code>	Random seed.
<code>backend</code>	"cpu", "cuda", or "metal".
<code>n.cores</code>	Number of CPU cores.
<code>graph_mode</code>	UMAP graph mode, passed unchanged to <code>umap()</code> . The standard "fuzzy" graph is the default.
<code>keep_knn</code>	Retain reference KNN output.
<code>verbose</code>	Print optimizer progress.
<code>...</code>	Additional optimizer arguments passed to <code>tsne()</code> .

Value

A `fastEmbedR_landmark_model`.

Examples

```

x <- scale(as.matrix(iris[, 1:4]))
selection <- select_landmarks(x, 0.5, seed = 1)
model <- fit_landmark_model(
  x, selection,
  method = "umap", n_neighbors = 10,
  graph_mode = "fuzzy", seed = 1
)

```

graph_cluster	<i>Native graph community detection</i>
---------------	---

Description

graph_cluster() clusters a compact fastEmbedR_graph with a native multilevel Louvain implementation, a Leiden local-move/refine/aggregate implementation, or the Pons-Latapy Walktrap algorithm. Louvain and Leiden have CPU, CUDA, and Metal implementations. Walktrap is CPU-only. No Python, igraph, cuGraph, or external clustering function is called at run time.

Usage

```
graph_cluster(
  graph,
  method = c("leiden", "louvain", "walktrap"),
  backend = NULL,
  resolution = 1,
  n_iterations = 10L,
  n_runs = 1L,
  steps = 4L,
  seed = 1L
)
```

Arguments

graph	A fastEmbedR_graph, or an edge-list list containing from, to, weight, and n_vertices.
method	"leiden", "louvain", or "walktrap". Walktrap is the package's true random-walk community method.
backend	Execution backend. "cuda" and "metal" are available for Louvain and Leiden only and fail explicitly if the requested accelerator was not compiled or is unavailable. They never fall back to CPU.
resolution	Positive modularity resolution for Louvain and Leiden. Walktrap uses its reference resolution of 1.
n_iterations	Maximum local-moving passes for Louvain and Leiden.
n_runs	Independent seeded Louvain/Leiden runs; the highest-modularity result is returned.
steps	Random-walk length for Walktrap.
seed	Reproducible random seed for Louvain and Leiden.

Value

A fastEmbedR_graph_cluster containing one-based membership, modularity, method, implementation, and run metadata.

References

Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E. Fast unfolding of communities in large networks. J Stat Mech. 2008;2008:P10008.

Traag VA, Waltman L, van Eck NJ. From Louvain to Leiden: guaranteeing well-connected communities. Sci Rep. 2019;9:5233.

Pons P, Latapy M. Computing communities in large networks using random walks. J Graph Algorithms Appl. 2006;10:191-218.

Examples

```
x <- scale(as.matrix(iris[, 1:4]))
graph <- knn_graph(x, k = 15)
clusters <- graph_cluster(graph, method = "leiden", seed = 1)
table(clusters$membership)
```

knn_graph

Build a compact nearest-neighbor graph

Description

knn_graph() converts a data matrix, a fastEmbedR_embedding, or a precomputed KNN object into a native undirected edge graph. When neighbors must be computed, the function uses fastEmbedR's existing native KNN route; it does not expose a second nearest-neighbor tuning API.

Usage

```
knn_graph(
  x,
  k = 30L,
  backend = NULL,
  metric = c("euclidean", "cosine", "correlation"),
  weight = c("snn", "distance", "binary"),
  mutual = FALSE,
  prune = 0,
  n.cores = 1L
)
```

Arguments

x	Numeric data matrix, a fastEmbedR_embedding, a list containing indices and distances, or an existing fastEmbedR_graph.
k	Number of non-self neighbors retained.
backend	KNN-search backend used only when x is data: "cpu", "cuda", or "metal".
metric	Distance metric used only when KNN must be computed: "euclidean", "cosine", or "correlation".

weight	Edge weighting: shared-neighbor Jaccard ("snn"), inverse distance ("distance"), or unit weights ("binary").
mutual	Keep only reciprocal KNN edges.
prune	Remove edges whose final weight is less than or equal to this non-negative threshold.
n.cores	CPU cores used by KNN search and graph construction.

Value

A compact fastEmbedR_graph list with from, to, weight, n_vertices, and construction metadata.

References

Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E. Fast unfolding of communities in large networks. J Stat Mech. 2008;2008:P10008.

Examples

```
x <- scale(as.matrix(iris[, 1:4]))
graph <- knn_graph(x, k = 15, weight = "snn")
graph
```

landmark_tsne	<i>Landmark t-SNE with fixed-reference transform</i>
---------------	--

Description

landmark_tsne() embeds a subset of observations with `tsne()`, then places the remaining observations with `transform_tsne()`. Projection KNN searches only the fixed landmark reference: CPU uses native HNSW, Metal uses native exact or recall-tuned IVF-Flat, and CUDA keeps native exact or IVF-Flat results resident on the device.

Usage

```
landmark_tsne(
  data,
  landmarks = TRUE,
  reference_method = c("tsne"),
  n_neighbors = NULL,
  perplexity = NULL,
  affinity_support = c("standard", "compact"),
  n_components = 2L,
  standardize = TRUE,
  pca_dims = NULL,
  seed = 4L,
  backend = NULL,
  transform_k = NULL,
```

```

transform_perplexity = 5,
transform_iter = 250L,
transform_early_exaggeration_iter = 0L,
transform_n_negatives = NULL,
initialization = c("median", "weighted", "random"),
keep_knn = FALSE,
verbose = FALSE,
n.cores = NULL,
...
)

```

Arguments

<code>data</code>	Numeric matrix/data frame with observations in rows.
<code>landmarks</code>	TRUE for an automatic subset, a fraction such as 0.5, a landmark count, or explicit row indices.
<code>reference_method</code>	Kept for compatibility. Only "tsne" is accepted in the cleaned package API.
<code>n_neighbors</code>	Number of non-self neighbors used to embed the landmark reference set. If NULL, it follows the same neighbor policy as <code>tsne()</code> : <code>ceiling(3 * perplexity)</code> under standard support.
<code>perplexity</code>	t-SNE perplexity for the landmark reference embedding. If NULL, the optimizer chooses a safe value from the reference KNN width and sample size.
<code>affinity_support</code>	Affinity candidate-neighborhood policy. The default "standard" uses <code>ceiling(3 * perplexity)</code> non-self neighbors so the Gaussian bandwidth search can produce non-uniform conditional probabilities. "compact" retains the older <code>ceiling(perplexity)</code> -neighbor approximation for explicit speed/memory experiments; it is not equivalent to conventional sparse t-SNE.
<code>n_components</code>	Output dimensionality, from 1 to 3. Dimensions other than two use CPU exact repulsion; the current Metal and CUDA interpolation/FFT optimizers support only 2L.
<code>standardize</code>	Center and scale columns before landmark selection and neighbor search. Unlike <code>tsne()</code> , landmark t-SNE defaults to TRUE.
<code>pca_dims</code>	Optional PCA dimension before KNN.
<code>seed</code>	Random seed.
<code>backend</code>	Execution backend: "cpu", "cuda", or "metal".
<code>transform_k</code>	Number of landmark neighbors used to place non-landmarks.
<code>transform_perplexity</code>	Perplexity used by <code>transform_tsne()</code> .
<code>transform_iter</code>	Number of normal transform iterations. Use 0 for projection-only landmarking with no transform refinement.
<code>transform_early_exaggeration_iter</code>	Number of transform early exaggeration iterations.

<code>transform_n_negatives</code>	Number of sampled reference negatives used by <code>transform_tsne()</code> on large landmark sets. GPU sampled transform repulsion is native and experimental for large reference sets.
<code>initialization</code>	Initial placement for transformed observations.
<code>keep_knn</code>	If TRUE, retain KNN matrices in the returned object.
<code>verbose</code>	Print optimizer progress.
<code>n.cores</code>	Number of CPU cores used by CPU KNN and CPU transform optimization. Native GPU stages ignore this argument.
<code>...</code>	Additional low-level parameters passed to <code>tsne_knn()</code> .

Value

A `fastEmbedR_embedding` object.

Examples

```
fit <- landmark_tsne(
  as.matrix(iris[, 1:4]),
  landmarks = 0.5, perplexity = 5,
  early_exaggeration_iter = 5, n_iter = 10,
  transform_iter = 5, seed = 1
)
plot(fit, labels = iris$Species)
```

<code>landmark_umap</code>	<i>Run landmark UMAP from a data matrix</i>
----------------------------	---

Description

`landmark_umap()` embeds a landmark subset with `umap()` and projects the remaining observations against the fixed landmark embedding using query-to-reference KNN, local affine placement, and optional fixed-reference refinement. It is an explicit landmark approximation: the UMAP objective and parameters for the landmark subset are unchanged.

Usage

```
landmark_umap(
  data,
  landmarks = 0.5,
  n_neighbors = NULL,
  n_components = 2L,
  standardize = TRUE,
  pca_dims = NULL,
  seed = 4L,
  backend = NULL,
  transform_k = NULL,
```

```

    n.cores = NULL,
    keep_knn = FALSE,
    graph_mode = c("fuzzy", "binary"),
    verbose = FALSE
)

```

Arguments

<code>data</code>	Numeric matrix/data frame, or a list containing KNN indices and distances.
<code>landmarks</code>	TRUE for an automatic subset, a fraction such as 0.5, a landmark count, or explicit row indices.
<code>n_neighbors</code>	Number of non-self neighbors. NULL chooses the package default for the data size.
<code>n_components</code>	Output dimensionality. The CPU backend supports positive dimensions, including three-dimensional embeddings. The current Metal and CUDA optimizers support only 2L.
<code>standardize</code>	Center and scale columns before landmark selection and neighbor search. Unlike <code>umap()</code> , landmark UMAP defaults to TRUE.
<code>pca_dims</code>	Optional PCA dimension before KNN.
<code>seed</code>	Random seed.
<code>backend</code>	Execution backend: "cpu", "cuda", or "metal". CPU KNN uses package-native HNSW. Metal uses package-native exact or recall-tuned IVF-Flat search. CUDA uses direct FAISS GPU exact search below 100,000 rows and direct RAPIDS cuVS IVF-Flat above that threshold; the KNN result stays on the device through graph construction and optimization. GPU requests must resolve to a real native backend; the package does not relabel CPU work as GPU.
<code>transform_k</code>	Number of landmark neighbors used to project non-landmark observations. Defaults to <code>n_neighbors</code> .
<code>n.cores</code>	Requested CPU core count. For matrix input, the value is passed to native CPU KNN and CPU UMAP; UMAP currently uses at most four workers and records both requested and effective values. NULL uses one KNN worker and a size-aware UMAP default from one to four workers. Native GPU stages ignore this argument.
<code>keep_knn</code>	Keep KNN matrices in the returned object.
<code>graph_mode</code>	Graph weighting mode passed unchanged to the reference <code>umap()</code> fit. "fuzzy" (the default) uses standard UMAP fuzzy graph weights; "binary" uses a symmetric unit-weight sensitivity graph. The fixed-reference projection algorithm is shared, but the fitted reference layout and its recorded metadata retain the selected graph mode.
<code>verbose</code>	Print progress.

Value

A `fastEmbedR_embedding` object containing the full layout, landmark and query indices, timings, and resolved parameters.

Examples

```
x <- scale(as.matrix(iris[1:60, 1:4]))
fit <- landmark_umap(x, landmarks = 0.5, n_neighbors = 10, seed = 1)
plot(fit)
```

pca

*Backend-native truncated PCA***Description**

`pca()` computes principal component scores with a backend-native truncated decomposition. CPU uses a package-native float32 blocked randomized SVD (RSVD). Metal uses a package-native float32 block-subspace TSVD whose large matrix products are executed with Metal Performance Shaders while data and work buffers remain resident in unified GPU memory. CUDA uses native RAPIDS RAFT TSVD with float32 input, score, and loading buffers. CUDA requests fail explicitly when RAFT TSVD support is unavailable; they never fall back to CPU PCA.

Usage

```
pca(
  x,
  ncomp = 2L,
  xtest = NULL,
  center = TRUE,
  scale = FALSE,
  backend = NULL,
  n.cores = 1L,
  seed = 4L,
  tsne_init = FALSE
)
```

Arguments

<code>x</code>	Numeric matrix/data frame or <code>float::float32</code> matrix with observations in rows.
<code>ncomp</code>	Number of principal components.
<code>xtest</code>	Optional test/query matrix with the same columns as <code>x</code> . Projected coordinates are returned in <code>scores_test</code> .
<code>center</code>	If <code>TRUE</code> , mean-center columns before decomposition.
<code>scale</code>	If <code>TRUE</code> , scale centered columns to unit sample standard deviation before decomposition.
<code>backend</code>	PCA backend: "cpu", "cuda", or "metal".
<code>n.cores</code>	Positive integer CPU core limit. It controls the linked BLAS/OpenMP numerical kernels when <code>backend = "cpu"</code> and is ignored by Metal and CUDA.

seed	Random seed for backends that use a Gaussian subspace sketch. The native RAFT covariance-eigensolver route records this value for provenance but does not consume random numbers.
tsne_init	If TRUE, add tsne_init to the returned PCA object. This matrix is centered and rescaled so its largest component standard deviation is 1e-4, ready to pass as Y_init to tsne() or tsne_knn() .

Details

CPU matrix products and factorizations use the BLAS/LAPACK linked to R. Set `n.cores` to control their CPU core limit. The requested value is applied temporarily through standard BLAS/OpenMP environment variables and, when installed, `RhpcBLASctl`; the prior process settings are restored after the call. A single-threaded BLAS can still report one effective thread. With `float::float32` input, native Metal and CUDA preprocessing and the returned scores/loadings also remain `float32`. The API intentionally has no decomposition method menu and does not call Python. Explicit unavailable GPU requests fail instead of being reported as GPU work.

Value

A `fastEmbedR_pca` list with scores, loadings, singular_values, centering/scaling vectors, backend metadata, and decomposition metadata. When `tsne_init = TRUE`, the list also contains `tsne_init`; when `xtest` is supplied, it also contains `scores_test`. Metal and CUDA preserve `float::float32` scores, loadings, initialization, and compatible test projections when the input is `float32`. CPU results also record `n.cores_requested`, `n.cores_effective`, and `core_control`.

Examples

```
fit <- pca(
  as.matrix(iris[, 1:4]),
  ncomp = 2,
  n.cores = 2,
  seed = 1,
  tsne_init = TRUE
)
plot(fit$scores, pch = 21, bg = iris$Species)
plot(fit$tsne_init, pch = 21, bg = iris$Species)
```

```
precompute_knn
```

```
Precompute native nearest neighbors
```

Description

`precompute_knn()` exposes the same package-native nearest-neighbor search used internally by [umap\(\)](#) and [tsne\(\)](#). The search algorithm is chosen by `fastEmbedR` for the requested backend and is deliberately not a user parameter.

Usage

```
precompute_knn(
  data,
  k = 30L,
  metric = c("euclidean", "cosine", "correlation", "inner_product"),
  backend = NULL,
  n.cores = NULL
)
```

Arguments

<code>data</code>	Numeric matrix, numeric data frame, or a <code>float::float32</code> matrix with observations in rows.
<code>k</code>	Number of non-self nearest neighbors to return.
<code>metric</code>	Distance metric: "euclidean", "cosine", "correlation", or "inner_product". Raw inner product is available only in CUDA builds that support it.
<code>backend</code>	Search backend: "cpu", "cuda", or "metal".
<code>n.cores</code>	Number of CPU cores. Native GPU backends ignore this argument.

Details

CPU search uses the package-native recall-tuned HNSW implementation. Metal uses native exact search for small inputs and recall-tuned IVF-Flat for larger inputs. CUDA uses package-native direct FAISS GPU exact search below 100,000 observations and direct RAPIDS cuVS IVF-Flat above that threshold. The internal recall target is 0.99.

The CUDA result remains on the GPU and can be passed directly to `umap_knn()` or `tsne_knn()` with `backend = "cuda"`. CPU and Metal results contain one-based indices and distances matrices. Every result excludes the observation itself. An unavailable requested backend raises an error; no CPU fallback is reported as GPU work.

Value

A `fastEmbedR_knn` object. Host results contain indices and distances; CUDA results additionally inherit from `fastEmbedR_gpu_knn` and own device-resident index and distance buffers.

Examples

```
x <- scale(as.matrix(iris[, 1:4]))
knn <- precompute_knn(x, k = 15, backend = "cpu", n.cores = 2)
layout <- umap_knn(knn, backend = "cpu", seed = 1)
```

```
precompute_query_knn
```

Precompute query-to-reference nearest neighbors

Description

`precompute_query_knn()` searches a fixed reference matrix for every row of a query matrix. It uses the same package-native backend family and routing policy as `precompute_knn()`, but does not compute unnecessary reference-to-reference or query-to-query neighbors.

Usage

```
precompute_query_knn(
  reference,
  query,
  k = 30L,
  metric = c("euclidean", "cosine", "correlation", "inner_product"),
  backend = NULL,
  n.cores = NULL
)
```

Arguments

<code>reference</code>	Reference observations in rows.
<code>query</code>	Query observations in rows and the same feature space as <code>reference</code> .
<code>k</code>	Number of non-self nearest neighbors to return.
<code>metric</code>	Distance metric: "euclidean", "cosine", "correlation", or "inner_product". Raw inner product is available only in CUDA builds that support it.
<code>backend</code>	Search backend: "cpu", "cuda", or "metal".
<code>n.cores</code>	Number of CPU cores. Native GPU backends ignore this argument.

Details

CPU uses the native recall-tuned HNSW reference-query path. Metal routes between a native query-only exact kernel and recall-tuned IVF-Flat from the estimated reference-query distance workload. CUDA uses direct FAISS GPU exact search below 100,000 reference rows and direct RAPIDS cuVS IVF-Flat above that threshold. CUDA results remain device-resident for direct consumption by landmark UMAP and t-SNE transformations.

Value

A `fastEmbedR_knn` object with one row per query observation and one-based indices into reference.

Examples

```
x <- scale(as.matrix(iris[, 1:4]))
ref <- x[1:100, , drop = FALSE]
qry <- x[101:150, , drop = FALSE]
knn <- precompute_query_knn(ref, qry, k = 10, backend = "cpu")
```

prepare_tsne_knn	<i>Precompute reusable t-SNE KNN state</i>
------------------	--

Description

prepare_tsne_knn() strips self-neighbors, trims to the requested non-self width, and stores compact KNN matrices once. Pass the returned object to [tsne_knn\(\)](#) for repeated seeds or backend comparisons without repeating KNN normalization/materialization.

Usage

```
prepare_tsne_knn(
  indices,
  distances = NULL,
  n_neighbors = NULL,
  perplexity = NULL,
  affinity_support = c("standard", "compact")
)
```

Arguments

indices	A list containing KNN indices and distances, or an integer KNN index matrix.
distances	Numeric KNN distance matrix matching indices. Leave NULL when indices is a KNN list.
n_neighbors	Optional number of non-self neighbor columns to use from the supplied KNN graph. This lets you compute a wide KNN once and reuse its first columns for comparable tests. If omitted, affinity_support determines the width. An explicit value must be at least the width implied by affinity_support.
perplexity	t-SNE perplexity. If NULL, the optimizer chooses a safe value from the supplied KNN width and sample size.
affinity_support	Affinity candidate-neighborhood policy. "standard" (default) uses ceiling(3 * perplexity) non-self neighbors, matching the conventional sparse t-SNE support rule. "compact" uses only ceiling(perplexity) neighbors and is an explicit approximation with nearly uniform conditional probabilities when the support equals the target perplexity.

Value

A prepared t-SNE KNN object.

Examples

```
x <- scale(as.matrix(iris[, 1:4]))
d <- as.matrix(stats::dist(x))
diag(d) <- Inf
k <- 15L
idx <- t(apply(d, 1L, order))[, seq_len(k), drop = FALSE]
dst <- matrix(d[cbind(rep(seq_len(nrow(d)), each = k), as.vector(t(idx)))],
  nrow = nrow(d), byrow = TRUE
)
prep <- prepare_tsne_knn(idx, dst, perplexity = 5)
y1 <- tsne_knn(prepare, seed = 1, early_exaggeration_iter = 5, n_iter = 10)
y2 <- tsne_knn(prepare, seed = 2, early_exaggeration_iter = 5, n_iter = 10)
```

prepare_umap_knn	<i>Precompute reusable UMAP graph state from KNN</i>
------------------	--

Description

prepare_umap_knn() builds the CSR graph once from a KNN object or index and distance matrices. The result can be passed to [umap_knn\(\)](#) or [embed_knn\(\)](#) repeatedly with different seeds without rebuilding the graph or recomputing epochs_per_sample.

Usage

```
prepare_umap_knn(
  indices,
  distances = NULL,
  backend = NULL,
  n.cores = NULL,
  graph_mode = c("fuzzy", "binary")
)
```

Arguments

indices	Integer matrix of nearest-neighbor indices, one row per observation, or a KNN object containing indices and distances. One-based and zero-based indices are accepted. A self-neighbor first column is removed automatically.
distances	Numeric matrix matching indices. Leave as NULL when indices is a KNN object.
backend	Execution backend: "cpu", "cuda", or "metal".
n.cores	Requested CPU core count. CPU UMAP currently uses at most four workers and records both requested and effective values. NULL selects a size-aware default from one to four workers. Native GPU backends ignore this argument.
graph_mode	Graph weighting mode. "fuzzy" (the default) uses standard UMAP fuzzy graph weights. "binary" uses a symmetric unit-weight sensitivity graph.

Value

A prepared UMAP object containing the KNN, CSR graph, and resolved UMAP graph schedule.

Examples

```
x <- scale(as.matrix(iris[, 1:4]))
d <- as.matrix(stats::dist(x))
diag(d) <- Inf
k <- 15L
idx <- t(apply(d, 1L, order))[, seq_len(k), drop = FALSE]
dst <- matrix(d[cbind(rep(seq_len(nrow(d)), each = k), as.vector(t(idx)))],
  nrow = nrow(d), byrow = TRUE
)
prep <- prepare_umap_knn(idx, dst)
y1 <- umap_knn(prepare, seed = 1)
y2 <- umap_knn(prepare, seed = 2)
```

```
print.fastEmbedR_knn
```

Print a precomputed nearest-neighbor object

Description

Print a precomputed nearest-neighbor object

Usage

```
## S3 method for class 'fastEmbedR_knn'
print(x, ...)
```

Arguments

<code>x</code>	A fastEmbedR_knn object.
<code>...</code>	Unused.

Value

`x`, invisibly.

```
print.fastEmbedR_umap_initialization
```

Print a reusable UMAP initialization

Description

Print a reusable UMAP initialization

Usage

```
## S3 method for class 'fastEmbedR_umap_initialization'
print(x, ...)
```

Arguments

x	A fastEmbedR_umap_initialization object.
...	Unused.

Value

x, invisibly.

```
project_landmark_model
```

Project observations into a fitted landmark embedding

Description

project_landmark_model() computes query-to-reference KNN with the same native backend policy used by the full embedding functions, then applies the method-specific fixed-reference transform. Landmark coordinates remain fixed.

Usage

```
project_landmark_model(
  model,
  data,
  query_indices = NULL,
  transform_k = NULL,
  refinement_epochs = 50L,
  transform_perplexity = 5,
  transform_iter = 250L,
  transform_early_exaggeration_iter = 0L,
  transform_n_negatives = NULL,
  initialization = c("median", "weighted", "random"),
```

```

    keep_knn = FALSE,
    n.cores = NULL,
    verbose = FALSE,
    ...
  )

```

Arguments

model	A model returned by <code>fit_landmark_model()</code> .
data	The original full data matrix for a model fitted on a strict landmark subset, or a matrix containing only new query observations in the same feature space. A model fitted with all training rows as references always treats data as new query observations.
query_indices	Optional rows of data to project.
transform_k	Number of reference neighbors.
refinement_epochs	Fixed-reference UMAP refinement epochs.
transform_perplexity	Perplexity of the t-SNE transform.
transform_iter	Number of t-SNE transform iterations.
transform_early_exaggeration_iter	Transform exaggeration iterations.
transform_n_negatives	Number of reference negatives for t-SNE.
initialization	t-SNE query initialization.
keep_knn	Retain query-to-reference KNN output.
n.cores	Number of CPU cores.
verbose	Print optimizer progress.
...	Low-level t-SNE transform controls.

Value

A `fastEmbedR_embedding`. For a strict landmark-subset model, passing the original full matrix reassembles layout in original row order. Otherwise layout contains only the supplied query rows. The returned parameters record `projection_scope` as "original_reconstruction" or "held_out_query".

Examples

```

x <- scale(as.matrix(iris[, 1:4]))
selection <- select_landmarks(x, 0.5, seed = 1)
model <- fit_landmark_model(
  x, selection,
  method = "umap", n_neighbors = 10, seed = 1
)
fit <- project_landmark_model(model, x, refinement_epochs = 2)

```

select_landmarks	<i>Select representative landmark observations</i>
------------------	--

Description

select_landmarks() separates row indices into a representative landmark reference and its complementary query set. It does not embed or project the data.

Usage

```
select_landmarks(data, landmarks = TRUE, seed = 4L, n.cores = NULL)
```

Arguments

data	Numeric matrix, data frame, or float::float32 matrix.
landmarks	TRUE for an automatic count, a fraction in (0, 1), a landmark count, or explicit row indices.
seed	Random seed used by projection-based landmark selection.
n.cores	Number of CPU cores used for float32 selection features.

Value

A fastEmbedR_landmark_selection object containing indices and query_indices.

Examples

```
x <- scale(as.matrix(iris[, 1:4]))
selection <- select_landmarks(x, landmarks = 0.5, seed = 1)
selection$indices
```

transform_tsne	<i>Transform query points into an existing t-SNE embedding</i>
----------------	--

Description

transform_tsne() places query observations into a fixed reference t-SNE embedding. It follows a fixed-reference t-SNE transform design: query-to-reference affinities are computed from KNN distances, query points are initialized from nearby reference points, and optimization moves only the query points.

Usage

```
transform_tsne(
  reference_layout,
  knn = NULL,
  reference_data = NULL,
  new_data = NULL,
  k = NULL,
  perplexity = 5,
  initialization = c("median", "weighted", "random"),
  Y_init = NULL,
  n_iter = 250L,
  early_exaggeration_iter = 0L,
  learning_rate = 0.1,
  early_exaggeration = 4,
  exaggeration = 1.5,
  initial_momentum = 0.8,
  final_momentum = 0.8,
  max_grad_norm = 0.25,
  max_step_norm = Inf,
  n_negatives = NULL,
  exact_repulsion_threshold = 4096L,
  n.cores = NULL,
  seed = 4L,
  backend = NULL,
  verbose = FALSE
)
```

Arguments

reference_layout	Numeric reference embedding matrix, or a fastEmbedR_embedding object.
knn	Optional query-to-reference KNN list with indices and distances.
reference_data	Reference observations in the same preprocessing space used to fit reference_layout. Required only when knn is NULL.
new_data	Query observations in the same preprocessing space as reference_data. Required only when knn is NULL.
k	Number of reference neighbors used for transform KNN. If NULL, uses at least 3 * perplexity, matching the usual t-SNE affinity width.
perplexity	Transform perplexity. The default is 5.
initialization	Initial query placement: "median" from reference neighbors, inverse-distance "weighted", or "random".
Y_init	Optional initial query embedding matrix. When supplied, it overrides initialization.
n_iter	Number of normal transform optimization iterations.
early_exaggeration_iter	Number of early-exaggeration iterations.

learning_rate	Transform learning rate.
early_exaggeration	Early exaggeration multiplier.
exaggeration	Normal transform exaggeration multiplier.
initial_momentum	Momentum used during early exaggeration.
final_momentum	Momentum used during normal optimization.
max_grad_norm	Maximum per-query gradient norm. Use Inf to disable.
max_step_norm	Maximum per-query step norm. Use Inf to disable.
n_negatives	Number of sampled reference points for repulsion when the reference set is larger than exact_repulsion_threshold. GPU sampled repulsion is native and experimental for large reference sets.
exact_repulsion_threshold	Use exact query-reference repulsion at or below this reference count.
n.cores	Number of CPU cores for the native optimizer.
seed	Random seed.
backend	Backend used for query KNN when knn is NULL; "metal" and "cuda" run their native fixed-reference transform optimizers when those backends were compiled. Explicit unavailable GPU requests fail; CPU work is never reported as Metal or CUDA.
verbose	Print native optimizer progress.

Value

A numeric matrix with one row per query observation.

Examples

```
reference <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), 4L, 2L, byrow = TRUE)
query_knn <- list(
  indices = matrix(c(1L, 2L, 3L, 4L), 2L, 2L, byrow = TRUE),
  distances = matrix(c(0.1, 0.2, 0.15, 0.25), 2L, 2L, byrow = TRUE)
)
query_layout <- transform_tsne(
  reference,
  knn = query_knn, perplexity = 0.5,
  n_iter = 2, exact_repulsion_threshold = 10, seed = 1
)
```

tsne

*Run native interpolation-based t-SNE from a data matrix***Description**

`tsne()` computes or reuses a KNN graph, then runs the package-native two-phase t-SNE optimizer. The default optimizer is CPU-native. Explicit `backend = "metal"` and `backend = "cuda"` requests use the matching package-native GPU optimizer when compiled and fail clearly if the requested backend is unavailable. The implementation follows published sparse-affinity and interpolation-based t-SNE methods and provides independent CPU, Metal, and CUDA kernels.

Usage

```
tsne(
  data,
  perplexity = NULL,
  affinity_support = c("standard", "compact"),
  n_components = 2L,
  init_data = NULL,
  Y_init = NULL,
  standardize = FALSE,
  pca_dims = NULL,
  metric = c("euclidean", "cosine", "correlation", "inner_product"),
  nn = NULL,
  seed = 4L,
  backend = NULL,
  keep_knn = FALSE,
  verbose = FALSE,
  n.cores = NULL,
  learning_rate = "auto",
  early_exaggeration_iter = NULL,
  early_exaggeration = "auto",
  n_iter = NULL,
  exaggeration = NULL,
  initial_momentum = 0.8,
  final_momentum = 0.8,
  max_step_norm = "auto",
  negative_gradient_method = "auto",
  record_costs = FALSE,
  auto_config = TRUE,
  ...
)
```

Arguments

<code>data</code>	Numeric matrix/data frame with observations in rows, or a list containing KNN indices and distances.
-------------------	--

perplexity	t-SNE perplexity. If NULL, uses the largest safe value up to 30 that is available for the input and selected affinity support.
affinity_support	Affinity candidate-neighborhood policy. The default "standard" uses <code>ceiling(3 * perplexity)</code> non-self neighbors so the Gaussian bandwidth search can produce non-uniform conditional probabilities. "compact" retains the older <code>ceiling(perplexity)</code> -neighbor approximation for explicit speed/memory experiments; it is not equivalent to conventional sparse t-SNE.
n_components	Output dimensionality, from 1 to 3. Dimensions other than two use CPU exact repulsion; the current Metal and CUDA interpolation/FFT optimizers support only 2L.
init_data	Optional original high-dimensional data matrix used only to compute PCA initialization with <code>tsne_pca_init()</code> . It is not used for neighbor search or optimization.
Y_init	Optional explicit initial layout. Use <code>tsne_pca_init()</code> to precompute and reuse a PCA initialization.
standardize	Center and scale columns before KNN. Defaults to FALSE so one-call results match a KNN object computed from the supplied matrix.
pca_dims	Optional PCA dimension before KNN.
metric	KNN distance metric for one-call matrix input: "euclidean", "cosine", "correlation", or "inner_product".
nn	Optional precomputed KNN output when data is a data matrix.
seed	Random seed.
backend	Execution backend: "cpu", "cuda", or "metal". CPU KNN uses package-native HNSW. Metal uses package-native exact or recall-tuned IVF-Flat search. CUDA uses direct FAISS GPU exact search below 100,000 rows and direct RAPIDS cuVS IVF-Flat above that threshold, then passes the device pointers directly to native CUDA t-SNE. Unsupported GPU requests fail clearly and are not relabelled CPU runs.
keep_knn	If TRUE, retain KNN matrices in the returned object.
verbose	Print optimizer progress.
n_cores	Number of CPU cores used by CPU KNN and CPU t-SNE optimization. NULL uses one CPU KNN worker and the package t-SNE thread option, which defaults to four. Native GPU optimizers ignore this argument.
learning_rate	Positive number or "auto". With "auto", the native optimizer uses <code>n / exaggeration</code> separately for each phase.
early_exaggeration_iter	Number of early-exaggeration iterations.
early_exaggeration	Early-exaggeration multiplier, or "auto" for the default t-SNE rule.
n_iter	Number of normal optimization iterations after early exaggeration.
exaggeration	Normal-phase exaggeration. NULL means 1.
initial_momentum	Momentum during early exaggeration.

<code>final_momentum</code>	Momentum during normal optimization.
<code>max_step_norm</code>	Maximum per-point update norm. "auto" uses the standard CPU limit and a tighter native Metal FFT-grid limit to avoid float32 outlier steps. Use NULL or NA to disable clipping.
<code>negative_gradient_method</code>	"auto", "exact", or "fft". On CPU, "auto" resolves to the native grid-FFT Fit-SNE-style negative-gradient approximation. Native GPU FFT/exact paths are used only when the corresponding compiled symbols are available; otherwise GPU requests fail clearly rather than falling back to CPU.
<code>record_costs</code>	If TRUE, compute diagnostic KL/cost traces.
<code>auto_config</code>	If TRUE, choose missing t-SNE settings with a native C++ opt-SNE-inspired policy. The policy uses <code>n / early_exaggeration</code> for "auto" learning rate, chooses missing iteration limits, and enables KLD-based early stopping only on CPU/small exact runs where the monitor is not prohibitively expensive. Explicit user-supplied values are respected.
<code>...</code>	Additional low-level parameters passed to <code>tsne_knn()</code> .

Details

The t-SNE API exposes the principal scientifically consequential optimizer controls: perplexity and candidate support, initialization, iteration counts, early and normal exaggeration, learning rate, momentum, clipping, and exact-versus-FFT repulsion. Setting `auto_config = FALSE` disables automatic iteration and stopping choices; explicit values always override automatic values. The matrix-input function deliberately does not expose a nearest-neighbor index type or tuning parameters: it uses the backend router with target recall 0.99. Supply an externally generated KNN object to `tsne_knn()` when search policy or affinity support must be controlled independently. The resolved settings are returned in `fit$parameters`.

Value

A `fastEmbedR_embedding` object.

Examples

```
fit <- tsne(
  as.matrix(iris[, 1:4]),
  perplexity = 5,
  early_exaggeration_iter = 5, n_iter = 10, seed = 1
)
plot(fit, labels = iris$Species)
```

tsne_knn

*Run native interpolation-based t-SNE from precomputed KNN***Description**

tsne_knn() is the direct KNN-input entry point for the native interpolation-based optimizer. It accepts either a list containing KNN indices and distances or separate KNN index and distance matrices. The supplied neighbors are used directly: this function performs no neighbor search or input scaling. PCA is computed only when `init_data` is supplied and `Y_init` is absent. The optimizer follows the sparse-affinity and interpolation-based t-SNE literature while using fastEmbedR's own R API and native kernels.

Usage

```
tsne_knn(
  indices,
  distances = NULL,
  n_neighbors = NULL,
  perplexity = NULL,
  affinity_support = c("standard", "compact"),
  n_components = 2L,
  init_data = NULL,
  Y_init = NULL,
  seed = 4L,
  verbose = FALSE,
  backend = NULL,
  n.cores = NULL,
  learning_rate = "auto",
  early_exaggeration_iter = NULL,
  early_exaggeration = "auto",
  n_iter = NULL,
  exaggeration = NULL,
  initial_momentum = 0.8,
  final_momentum = 0.8,
  max_step_norm = "auto",
  negative_gradient_method = "auto",
  record_costs = FALSE,
  auto_config = TRUE,
  ...
)
```

Arguments

<code>indices</code>	A list containing KNN indices and distances, or an integer KNN index matrix.
<code>distances</code>	Numeric KNN distance matrix matching indices. Leave NULL when indices is a KNN list.

n_neighbors	Optional number of non-self neighbor columns to use from the supplied KNN graph. This lets you compute a wide KNN once and reuse its first columns for comparable tests. If omitted, affinity_support determines the width. An explicit value must be at least the width implied by affinity_support.
perplexity	t-SNE perplexity. If NULL, the optimizer chooses a safe value from the supplied KNN width and sample size.
affinity_support	Affinity candidate-neighborhood policy. "standard" (default) uses ceiling(3 * perplexity) non-self neighbors, matching the conventional sparse t-SNE support rule. "compact" uses only ceiling(perplexity) neighbors and is an explicit approximation with nearly uniform conditional probabilities when the support equals the target perplexity.
n_components	Output dimensionality, from 1 to 3. Dimensions other than two use CPU exact repulsion; the current Metal and CUDA interpolation/FFT optimizers support only 2L.
init_data	Optional original high-dimensional data matrix used only to compute PCA initialization for KNN-input runs. It is not used for neighbor search or optimization.
Y_init	Optional explicit initial layout. Use tsne_pca_init() to precompute and reuse a PCA initialization.
seed	Random seed.
verbose	Print optimizer progress.
backend	Optimizer backend: "cpu", "cuda", or "metal". This selects affinity construction and layout optimization only; it does not trigger a nearest-neighbor search. Host KNN matrices can be consumed by any compiled backend. A CUDA-resident KNN object can be consumed directly only by the CUDA backend. Unavailable GPU requests fail without CPU fallback.
n_cores	Number of CPU cores used for CPU affinity construction and t-SNE optimization. NULL uses the package t-SNE thread option, which defaults to four. No KNN search is performed. Metal and CUDA optimizers ignore this argument.
learning_rate	Positive number or "auto". With "auto", the native optimizer uses n / exaggeration separately for each phase.
early_exaggeration_iter	Number of early-exaggeration iterations.
early_exaggeration	Early-exaggeration multiplier, or "auto" for the default t-SNE rule.
n_iter	Number of normal optimization iterations after early exaggeration.
exaggeration	Normal-phase exaggeration. NULL means 1.
initial_momentum	Momentum during early exaggeration.
final_momentum	Momentum during normal optimization.
max_step_norm	Maximum per-point update norm. "auto" uses the standard CPU limit and a tighter native Metal FFT-grid limit to avoid float32 outlier steps. Use NULL or NA to disable clipping.

`negative_gradient_method` "auto", "exact", or "fft". On CPU, "auto" resolves to the native grid-FFT FIt-SNE-style negative-gradient approximation. Native GPU FFT/exact paths are used only when the corresponding compiled symbols are available; otherwise GPU requests fail clearly rather than falling back to CPU.

`record_costs` If TRUE, compute diagnostic KL/cost traces.

`auto_config` If TRUE, choose missing t-SNE settings with a native C++ opt-SNE-inspired policy. The policy uses `n / early_exaggeration` for "auto" learning rate, chooses missing iteration limits, and enables KLD-based early stopping only on CPU/small exact runs where the monitor is not prohibitively expensive. Explicit user-supplied values are respected.

... Additional low-level optimizer controls, including `theta` and `min_gain`.

Value

An embedding matrix with settings stored in `attr(layout, "fastEmbedR_config")`. Float32 KNN distances, including a CUDA-resident KNN object, produce a `float::float32` layout; host double distances produce a standard R double matrix. Native optimization uses float32 in both cases. The exact returned representation is recorded in `attr(layout, "precision")` and in `fastEmbedR_config$output_p`.

Examples

```
x <- scale(as.matrix(iris[, 1:4]))
d <- as.matrix(stats::dist(x))
diag(d) <- Inf
k <- 15L
idx <- t(apply(d, 1L, order))[, seq_len(k), drop = FALSE]
dst <- matrix(d[cbind(rep(seq_len(nrow(d)), each = k), as.vector(t(idx)))],
  nrow = nrow(d), byrow = TRUE
)
layout <- tsne_knn(idx, dst,
  init_data = x, perplexity = 5,
  early_exaggeration_iter = 5, n_iter = 10
)
```

tsne_pca_init

Compute or reuse a PCA initialization for t-SNE

Description

`tsne_pca_init()` creates the small-scale PCA initialization used by `tsne()` and `tsne_knn()`. Supplying `cache_file` stores the result as an RDS file; later calls with the same path reuse the saved matrix instead of recomputing PCA. This is useful when comparing several KNN backends with exactly the same initialization. For `backend = "cuda"`, `fastEmbedR` requires native RAPIDS RAFT TSVD support compiled into the CUDA backend and fails loudly if that backend is unavailable. Metal uses `fastEmbedR`'s native float32 block-subspace TSVD with native Metal centering/scaling, Metal Performance Shaders matrix products, and one resident GPU workspace. CPU uses `fastEmbedR`'s native float32 blocked RSVD implementation. The package does not call Python or reticulate for PCA initialization.

Usage

```
tsne_pca_init(
  data,
  n_components = 2L,
  seed = 4L,
  backend = NULL,
  n.cores = 1L,
  cache_file = NULL,
  force_recompute = FALSE
)
```

Arguments

<code>data</code>	Numeric matrix/data frame or <code>float::float32</code> matrix with observations in rows.
<code>n_components</code>	Output dimensionality, usually 2.
<code>seed</code>	Random seed used by the truncated PCA subspace sketch.
<code>backend</code>	Backend used for PCA when available: "cpu", "metal", or "cuda".
<code>n.cores</code>	Number of CPU cores used for CPU PCA initialization. Metal and CUDA ignore this argument.
<code>cache_file</code>	Optional .rds file path. If it exists and <code>force_recompute = FALSE</code> , the saved initialization is loaded and validated.
<code>force_recompute</code>	If TRUE, ignore any existing cache and recompute.

Value

An initialization matrix suitable for `Y_init`. Float32 input is returned as `float::float32` when the selected PCA backend preserves it.

Examples

```
init <- tsne_pca_init(as.matrix(iris[, 1:4]), seed = 1)
plot(init, pch = 21, bg = iris$Species)
```

umap

Run opinionated fixed-policy UMAP from data or precomputed KNN

Description

`umap()` is a small convenience wrapper. If data is already a list of KNN indices and distances, it calls `umap_knn()` directly. Otherwise it preprocesses the data, computes KNN once with fastEmbedR's native backend, and embeds from that KNN graph.

Usage

```
umap(
  data,
  n_neighbors = NULL,
  n_components = 2L,
  standardize = FALSE,
  pca_dims = NULL,
  metric = c("euclidean", "cosine", "correlation", "inner_product"),
  nn = NULL,
  seed = 4L,
  backend = NULL,
  n.cores = NULL,
  keep_knn = FALSE,
  graph_mode = c("fuzzy", "binary"),
  verbose = FALSE
)
```

Arguments

<code>data</code>	Numeric matrix/data frame, or a list containing KNN indices and distances.
<code>n_neighbors</code>	Number of non-self neighbors. NULL chooses the package default for the data size.
<code>n_components</code>	Output dimensionality. The CPU backend supports positive dimensions, including three-dimensional embeddings. The current Metal and CUDA optimizers support only 2L.
<code>standardize</code>	Center and scale columns before KNN when data is a matrix. Defaults to FALSE so one-call results match a KNN object computed from the supplied matrix.
<code>pca_dims</code>	Optional PCA dimension before KNN.
<code>metric</code>	KNN distance metric for one-call matrix input: "euclidean", "cosine", "correlation", or "inner_product".
<code>nn</code>	Optional precomputed KNN result when data is a matrix.
<code>seed</code>	Random seed.
<code>backend</code>	Execution backend: "cpu", "cuda", or "metal". CPU KNN uses package-native HNSW. Metal uses package-native exact or recall-tuned IVF-Flat search. CUDA uses direct FAISS GPU exact search below 100,000 rows and direct RAPIDS cuVS IVF-Flat above that threshold; the KNN result stays on the device through graph construction and optimization. GPU requests must resolve to a real native backend; the package does not relabel CPU work as GPU.
<code>n.cores</code>	Requested CPU core count. For matrix input, the value is passed to native CPU KNN and CPU UMAP; UMAP currently uses at most four workers and records both requested and effective values. NULL uses one KNN worker and a size-aware UMAP default from one to four workers. Native GPU stages ignore this argument.
<code>keep_knn</code>	Keep KNN matrices in the returned object.

graph_mode	Graph weighting mode. "fuzzy" (the default) uses the standard UMAP fuzzy graph. "binary" uses a symmetric unit-weight graph as an adjacency-only sensitivity approximation; it is not standard UMAP or necessarily faster.
verbose	Print progress.

Details

`umap()` is an opinionated high-throughput UMAP implementation rather than a drop-in interface for every UMAP hyperparameter. The public matrix-input API exposes neighborhood size, distance metric, graph weighting, output dimension, preprocessing, backend, seed, and CPU thread count. In the current release, epochs, minimum distance, spread, learning rate, repulsion strength, negative-sample rate, spectral iteration count, and asynchronous update mode are selected internally. The resolved values and rule names are stored in `fit$parameters`.

The default policy uses 500 epochs below 10,000 observations and 200 epochs for larger inputs, increasing to 300 for high-variability distance profiles; five negative samples, spread 1, and repulsion strength 1 are fixed. Minimum distance is normally 0.01 and becomes 0.1 only under the documented wide-shell profile; learning rate is correspondingly 1 or 1.25. Use `umap_init()` to precompute and reuse package-native initial coordinates, or `umap_knn()` to supply an externally controlled KNN graph. These controls are scientifically consequential; fixing them is a scope decision made to keep one tested objective and update schedule aligned across CPU, Metal, and CUDA, not a claim that the values are universally optimal. Arbitrary `min_dist`, `spread`, `epoch`, `learning-rate`, `negative-sampling`, or `optimizer-mode` sweeps should use a general-purpose UMAP implementation; `fastEmbedR` does not claim API or parameter interchangeability with those packages.

Value

A `fastEmbedR_embedding` object.

Examples

```
x <- scale(as.matrix(iris[, 1:4]))
fit <- umap(x, n_neighbors = 15, seed = 1)
plot(fit)
```

umap_init

Precompute a reusable UMAP initialization

Description

`umap_init()` separates UMAP neighbor search, graph construction, and initialization from stochastic optimization. The returned object can be passed directly to `umap_knn()` so repeated or cross-backend runs reuse the same KNN graph and initial coordinates.

Usage

```
umap_init(
  x,
  distances = NULL,
  n_neighbors = NULL,
  n_components = 2L,
  metric = c("euclidean", "cosine", "correlation", "inner_product"),
  backend = NULL,
  seed = 4L,
  n.cores = NULL,
  graph_mode = c("fuzzy", "binary")
)
```

Arguments

<code>x</code>	Numeric data, a KNN object, or an object returned by prepare_umap_knn() .
<code>distances</code>	Optional KNN distance matrix when <code>x</code> is an index matrix.
<code>n_neighbors</code>	Number of non-self neighbors when <code>x</code> is data. NULL uses the package's size-aware default.
<code>n_components</code>	Output dimensionality.
<code>metric</code>	Distance metric used only when KNN must be computed.
<code>backend</code>	Initialization backend: "cpu", "cuda", or "metal".
<code>seed</code>	Random seed.
<code>n.cores</code>	Number of CPU cores.
<code>graph_mode</code>	UMAP graph weighting. "fuzzy" is the standard default; "binary" requests the adjacency-only sensitivity graph.

Details

For the fuzzy graph, CPU initialization is computed from the prepared CSR graph, while native Metal or CUDA spectral kernels are used when requested and available. Binary initialization is computed from the same unit-weight CSR graph used by the optimizer, so its mathematical input is unchanged. Explicit GPU requests fail when the corresponding native backend is unavailable.

CUDA KNN input is materialized on the host only for this standalone diagnostic API. The ordinary one-call CUDA UMAP path remains device resident.

Value

A `fastEmbedR_umap_initialization` object with layout, prepared, resolved parameters, and component timings. Pass the complete object to [umap_knn\(\)](#) or use its layout component directly for diagnostics.

Examples

```
x <- scale(as.matrix(iris[, 1:4]))
initialization <- umap_init(
  x,
  n_neighbors = 15,
  backend = "cpu",
  graph_mode = "fuzzy",
  seed = 1
)
layout <- umap_knn(initialization, backend = "cpu", seed = 1)
plot(layout, pch = 21, bg = iris$Species)
```

umap_knn

*Run UMAP from precomputed nearest neighbors***Description**

Run UMAP from precomputed nearest neighbors

Usage

```
umap_knn(
  indices,
  distances = NULL,
  n_components = 2L,
  seed = 42L,
  verbose = FALSE,
  backend = NULL,
  n.cores = NULL,
  graph_mode = c("fuzzy", "binary")
)
```

Arguments

indices	Integer matrix of nearest-neighbor indices, one row per observation, or a KNN object containing indices and distances. One-based and zero-based indices are accepted. A self-neighbor first column is removed automatically.
distances	Numeric matrix matching indices. Leave as NULL when indices is a KNN object.
n_components	Output dimensionality. The CPU backend supports positive dimensions, including three-dimensional embeddings. The current Metal and CUDA optimizers support only 2L.
seed	Integer random seed.
verbose	Print native optimizer progress.
backend	Execution backend: "cpu", "cuda", or "metal".

<code>n.cores</code>	Requested CPU core count. CPU UMAP currently uses at most four workers and records both requested and effective values. NULL selects a size-aware default from one to four workers. Native GPU backends ignore this argument.
<code>graph_mode</code>	Graph weighting mode. "fuzzy" (the default) uses standard UMAP fuzzy graph weights. "binary" uses a symmetric unit-weight sensitivity graph.

Details

This KNN-input function exposes the same deliberately constrained optimizer policy as `umap()`. It reuses the supplied neighbors but still selects epochs, minimum distance, spread, learning rate, repulsion strength, negative-sample rate, spectral iterations, and backend update mode internally. The resolved configuration is attached as `attr(layout, "fastEmbedR_config")`. Pass an object from `umap_init()` to reuse both the prepared graph and package-native initialization. This API is intended for fixed-boundary comparisons and repeated seeds, not arbitrary UMAP hyperparameter sweeps.

Value

An embedding matrix with `nrow(indices)` rows and `n_components` columns. Float32 KNN input returns a `float::float32` layout; ordinary R input returns a numeric matrix. Resolved settings are stored in `attr(layout, "fastEmbedR_config")`.

Examples

```
x <- scale(as.matrix(iris[, 1:4]))
d <- as.matrix(stats::dist(x))
diag(d) <- Inf
k <- 10L
idx <- t(apply(d, 1L, order))[, seq_len(k), drop = FALSE]
dst <- matrix(
  d[cbind(rep(seq_len(nrow(d)), each = k), as.vector(t(idx)))],
  nrow = nrow(d), byrow = TRUE
)
layout <- umap_knn(idx, dst, seed = 1)
plot(layout, pch = 21, bg = iris$Species)
```

Index

embed_knn, 3
embed_knn(), 22
evaluate_embedding, 4

fastEmbedR_api, 6
fastEmbedR_backend, 6
fastEmbedR_backend(), 7
fastEmbedR_capabilities, 7
fastEmbedR_capabilities(), 6
fastEmbedR_embedding_methods, 8
fastEmbedR_graph_methods, 9
fit_landmark_model, 9
fit_landmark_model(), 25

graph_cluster, 11
graphics::plot(), 8

knn_graph, 12

landmark_tsne, 13
landmark_umap, 15

pca, 17
plot.fastEmbedR_embedding
 (fastEmbedR_embedding_methods),
 8
precompute_knn, 18
precompute_knn(), 20
precompute_query_knn, 20
prepare_tsne_knn, 21
prepare_umap_knn, 22
prepare_umap_knn(), 38
print.fastEmbedR_embedding
 (fastEmbedR_embedding_methods),
 8
print.fastEmbedR_graph
 (fastEmbedR_graph_methods), 9
print.fastEmbedR_graph_cluster
 (fastEmbedR_graph_methods), 9
print.fastEmbedR_knn, 23

print.fastEmbedR_umap_initialization,
 24
project_landmark_model, 24

select_landmarks, 26
select_landmarks(), 10

transform_tsne, 26
tsne, 29
tsne(), 9, 10, 13, 14, 18, 34
tsne_knn, 32
tsne_knn(), 15, 18, 19, 21, 31, 34
tsne_pca_init, 34
tsne_pca_init(), 30, 33

umap, 35
umap(), 6, 9, 10, 15, 16, 18, 40
umap_init, 37
umap_init(), 37, 40
umap_knn, 39
umap_knn(), 19, 22, 35, 37, 38