

Package: bixverse.gpu (via r-universe)

October 22, 2026

Title GPU-accelerated methods via cubeccl and WGPU for the bixverse with general methods such as k-means and correlations and a lot of methods specific for single cell analysis acceleration.

Version 0.6.1

Description Provides GPU-accelerated methods for the bixverse. These leverage Burn, cubeccl and wgpu under the hood to work on very different hardwares.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/rextendr/version 0.5.0

SystemRequirements Cargo (Rust's package manager), rustc >= 1.95, xz

Depends R (>= 4.2)

Suggests BiocNeighbors, bixverse.plots, knitr, quarto, tinytest

Imports bixverse (>= 0.8.0), checkmate, data.table, igraph, lifecycle, magrittr, manifoldsR (>= 0.3.0), Matrix, purrr, qs2, rextendr, S7, utils, zeallot

Additional_repositories <https://gregorlueg.r-universe.dev>

URL <https://gregorlueg.github.io/bixverse.gpu/>

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake libglpk-dev make libicu-dev libxml2-dev libssl-dev xz-utils libclang-dev

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

Date/Publication 2026-10-22 08:28:33 UTC

RemoteUrl <https://github.com/GregorLueg/bixverse.gpu>

RemoteRef main

RemoteSha f3cb01eb4f12c263c2e3f1069f7e2c917f399ac9

Contents

bbknn_gpu_sc	3
bonsai_gpu_sc	4
calculate_pca_gpu_sc	5
consensus_nmf_gpu_sc	6
fast_cluster_gpu_sc	8
fast_mnn_gpu_sc	9
find_clusters_gpu_sc	10
find_neighbours_gpu_sc	11
generate_gpu_knn_sc	13
generate_knn_graph_gpu	14
generate_seacells_gpu_sc	15
gpu_available	17
harmony_v2_gpu_sc	17
install_agent_skill_gpu	18
k_means_cluster_gpu	19
leiden_clustering_gpu	20
load_parametric_umap	21
mlx_available	22
nebula_gpu_sc	22
nmf_gpu_sc	24
nmf_k_sweep_gpu_sc	25
parametric_umap	27
params_kmeans_gpu	28
params_knn_gpu_defaults	29
params_nebula_gpu	30
params_nn_gpu	32
params_parametric_umap	34
params_sc_bbknn_gpu	35
params_sc_fast_cluster_gpu	36
params_sc_fastmnn_gpu	39
params_sc_harmony_v2_gpu	40
params_scrublet_gpu	43
params_tsne_gpu	44
params_umap_gpu	46
predict.ParametricUmapModel	47
rs_cor_gpu	48
rs_cov_gpu	49
rs_fast_mnn_gpu	49
rs_gpu_available	50
rs_gpu_knn	50
rs_gpu_plane_ops	51
rs_harmony_v2_gpu	51
rs_kmeans_gpu	52
rs_leiden_clustering_gpu	53
rs_parametric_umap	53
rs_parametric_umap_predict	54

rs_tsne_from_knn_gpu	55
rs_tsne_gpu	56
rs_umap_from_knn_gpu	57
rs_umap_gpu	58
save_parametric_umap	59
scenic_grn_sc_gpu	60
scrublet_gpu_sc	61
stabilised_nmf_gpu_sc	63
tsne_gpu	64
tsne_gpu_sc	65
umap_gpu	67
umap_gpu_sc	68

Index	71
--------------	-----------

bbknn_gpu_sc	<i>Run BBKNN on the GPU</i>
--------------	-----------------------------

Description

GPU counterpart of `bixverse::bbknn_sc()`, implementing the batch-balanced k-nearest neighbour algorithm from Polański, et al. One nearest neighbour index is built per batch and queried by every cell, so each cell gets `neighbours_within_batch` neighbours from every batch. The UMAP connectivity calculations that reduce spurious connections then run on the CPU, shared with the CPU implementation.

Usage

```
bbknn_gpu_sc(
  object,
  batch_column,
  no_neighbours_to_keep = 5L,
  embd_to_use = "pca",
  no_embd_to_use = NULL,
  bbknn_params = params_sc_bbknn_gpu(),
  seed = 42L,
  .verbose = TRUE
)
```

Arguments

<code>object</code>	SingleCells or SingleCellsSubset class from bixverse.
<code>batch_column</code>	String. The column with the batch information in the obs data of the class.
<code>no_neighbours_to_keep</code>	Integer. Maximum number of neighbours to keep from the BBKNN algorithm. Generating neighbours per batch can produce a lot of them, so this keeps the top <code>no_neighbours_to_keep</code> . Defaults to 5L.

embd_to_use	String. The embedding to use. Atm, the only option is "pca".
no_embd_to_use	Optional integer. Number of embedding dimensions to use. If NULL all will be used.
bbknn_params	List. Output of <code>params_sc_bbknn_gpu()</code> .
seed	Integer. Random seed.
.verbose	Boolean or integer. Controls verbosity and returns run times. FALSE -> quiet, TRUE or 1L -> normal verbosity, 2L -> detailed verbosity.

Details

Only the per-batch searches move to the device, and they are the part that scales with batch count: BBKNN builds one index per batch and queries each with all cells, so the work grows as $n_{\text{cells}} * n_{\text{batches}}$. With a handful of batches on a small object the CPU is fine. The GPU starts to matter once you have many samples.

Results match the CPU path exactly with `knn_method = "exhaustive"`, since both are exact and recompute distances against the same embedding. The approximate backends break ties differently and will not.

Value

The object with the added kNN matrix based on BBKNN and the graph based on the returned connectivities of the algorithm.

References

Polański, et al., Bioinformatics, 2020

bonsai_gpu_sc	<i>Build a Bonsai tree over the cells, Sanity on the GPU</i>
---------------	--

Description

GPU counterpart of `bixverse::bonsai_sc()`. Sanity, which turns the raw counts into posterior log fold changes with error bars and is most of the runtime on the CPU, runs on the WGPU backend. Genes are streamed through it in chunks and only the ones passing Bonsai's signal-to-noise filter are kept. The tree search and the layout run on the CPU, as they do in bixverse.

The device computes in f32, so the posteriors match the CPU run to within the resolution of Sanity's variance grid rather than bit for bit, and a gene sitting right at the signal-to-noise threshold can end up on the other side.

Usage

```
bonsai_gpu_sc(
  object,
  hvg = NULL,
  bonsai_params = bixverse::params_sc_bonsai(),
  .verbose = TRUE
)
```

Arguments

object	SingleCells or MetaCells class.
hvg	Optional integer. Restrict the candidate genes to these. Please provide 1-indexed genes here! If NULL, every gene in the object is a candidate.
bonsai_params	List. See bixverse::params_sc_bonsai() .
.verbose	Boolean or integer. Controls verbosity and returns run times. FALSE -> quiet, TRUE or 1L -> normal verbosity, 2L -> detailed verbosity.

Value

A BonsaiTree S3 object, see [bixverse::bonsai_sc\(\)](#).

References

de Groot, et al., Nat. Biotechnol., 2026; Breda, et al., Nat. Biotechnol., 2021.

calculate_pca_gpu_sc *GPU-accelerated PCA for single cell*

Description

This function will run a sparse SVD while running the large sparse matrix multiplications on GPU for improved speed. This also means you will have to provide the necessary VRAM for your data set. This version only works on the "rna" modality.

Usage

```
calculate_pca_gpu_sc(
  object,
  no_pcs,
  pca_params = bixverse::params_sc_pca(),
  hvg = NULL,
  seed = 42L,
  backend = c("wgpu", "mlx"),
  .verbose = TRUE
)
```

Arguments

object	SingleCells class
no_pcs	Integer. Number of PCs to calculate.
pca_params	Named list. Controls the parameters to be used for the PCA calculation which is single cell-specific, see params_sc_pca() . svd_solver = "randomised" picks the randomised SVD, anything else the exact block Lanczos.

hvg	Optional integer. If you want to provide your own HVG genes. Otherwise, the function will default to what is found in <code>bixverse::get_hvg()</code> . Please provide 1-indexed genes here! If you provide these, the internal HVG will be overwritten.
seed	Integer. Seed for the sketch or the Lanczos starting block.
backend	String. One of <code>c("wgpu", "mlx")</code> . "mlx" runs the SVD on Apple Silicon via MLX and needs a build with the MLX backend, see <code>mlx_available()</code> .
.verbose	Boolean or integer. Controls verbosity and returns run times. FALSE -> quiet, TRUE or 1L -> normal verbosity, 2L -> detailed verbosity.

Details

`svd_solver = "randomised"` runs the randomised SVD: fast, but approximate in the trailing PCs. Every other solver ("`auto`", "`exact`", "`covariance`") runs an exact block Lanczos, as there is no GPU covariance path. The default from `bixverse::params_sc_pca()` is "`auto`", so the default is exact and slower than the randomised solver.

Value

The function will add the PCA factors, loadings and singular values to the object cache in memory.

`consensus_nmf_gpu_sc` *Run consensus NMF on the GPU over single cell or meta cell data*

Description

GPU counterpart of `bixverse::consensus_nmf_sc()`. Runs `n_runs` HALS restarts on the device, pools their components, drops unstable ones by local density, k-means clusters the survivors and refits the partner factor against the per-cluster median.

Prefer this over `stabilised_nmf_gpu_sc()`, which picks the lowest-loss restart. Use `nmf_k_sweep_gpu_sc()` first if you do not already know `k`.

Usage

```
consensus_nmf_gpu_sc(
  object,
  k,
  cell_ids = NULL,
  gene_ids = NULL,
  preprocessing = "none",
  use_second_layer = TRUE,
  nmf_hals_params = bixverse::params_nmf_hals(),
  nmf_consensus_params = bixverse::params_nmf_consensus(),
  n_runs = 30L,
  seed = 42L,
  backend = c("wgpu", "mlx"),
  .verbose = TRUE
)
```

Arguments

object	SingleCells or MetaCells class from bixverse.
k	Integer. Number of latent factors. At least 2, at most 128, see NMF_GPU_MAX_RANK .
cell_ids	Optional character. Cell ids (or meta cell ids) to restrict the NMF to. If NULL, uses bixverse::get_cells_to_keep() for SingleCells and all meta cells for MetaCells.
gene_ids	Optional character. Gene ids to restrict the NMF to. If NULL, uses bixverse::get_hvg() on the object.
preprocessing	String. One of c("none", "sd", "sqrt_sd").
use_second_layer	Boolean. If TRUE, runs NMF on the normalised counts (recommended); if FALSE, on the raw counts.
nmf_hals_params	List, see bixverse::params_nmf_hals() . The nmf_init field is ignored, restarts always use random initialisation.
nmf_consensus_params	List, see bixverse::params_nmf_consensus() .
n_runs	Integer. Number of restarts. At least 2.
seed	Integer. Random seed for initialisation.
backend	String. One of c("wgpu", "mlx"). "mlx" runs the HALS solves on Apple Silicon via MLX and needs a build with the MLX backend, see mlx_available() .
.verbose	Boolean or integer. Controls verbosity. FALSE -> quiet, TRUE or 1L -> normal verbosity, 2L -> detailed verbosity.

Details

Only the restarts move to the GPU. The pooling, the density filter, the k-means and the silhouette all run on the CPU, shared with the bixverse implementation, so the speed-up tracks how much of the run the solves own.

The restart factors are dense and all held at once, so budget for n_runs times k times the cell count on top of the counts themselves.

If the density filter leaves fewer than k components, or a cluster comes out empty, the run errors rather than returning a partial answer. Raise density_threshold (2 switches the filter off) or increase n_runs.

Value

A ConsensusNmfResult object, the same class [bixverse::consensus_nmf_sc\(\)](#) returns.

References

Kotliar et al., eLife, 2019

fast_cluster_gpu_sc *Run fast clustering on a SingleCells object (GPU)*

Description

GPU counterpart of `bixverse::fast_cluster_sc()`. Coarsens the cells with k-means on the chosen embedding, builds a graph over the centroids (a kNN graph or the cell-mediated graph), runs Louvain or Leiden on it and propagates the communities back to the cells. With `n_seeds > 1` every resolution is run once per seed and stability statistics are returned.

Only the k-means coarsening (and the cell-mediated top-m search) run on the GPU. The centroid graph and the community detection stay on the CPU, so the speedup tracks how much of the run k-means owns. That share grows with the cell count and with `n_centroids`. There is no `km_type` argument: the GPU k-means is full-batch Lloyd's and has no mini-batch path.

Usage

```
fast_cluster_gpu_sc(
  object,
  embd_to_use = "pca",
  no_embd_to_use = NULL,
  resolutions = c(2, 1, 0.5),
  fc_params = params_sc_fast_cluster_gpu(),
  n_seeds = 1L,
  return_kmeans = FALSE,
  seed = 42L,
  backend = c("wgpu", "mlx"),
  n_centroids = lifecycle::deprecated(),
  snn = lifecycle::deprecated(),
  grid_search = lifecycle::deprecated(),
  no_seeds = lifecycle::deprecated(),
  .verbose = TRUE
)
```

Arguments

<code>object</code>	SingleCells or SingleCellsSubset class from bixverse.
<code>embd_to_use</code>	String. Embedding name. Defaults to "pca".
<code>no_embd_to_use</code>	Optional integer. Number of dimensions to keep.
<code>resolutions</code>	Numeric vector. Resolutions of the community detection.
<code>fc_params</code>	List. Output of <code>params_sc_fast_cluster_gpu()</code> .
<code>n_seeds</code>	Integer. Number of seeds per resolution. 1L runs a single pass without stability statistics.
<code>return_kmeans</code>	Boolean. Return the k-means assignments and centroids.
<code>seed</code>	Integer. Seed for reproducibility.

backend	String. One of c("wgpu", "mlx"). "mlx" runs the k-means on Apple Silicon via MLX and needs a build with the MLX backend, see mlx_available() .
n_centroids	[Deprecated] Use params_sc_fast_cluster_gpu(n_centroids =).
snn	[Deprecated] Use params_sc_fast_cluster_gpu(centroid_graph = list(knn_sym =)).
grid_search	[Deprecated] Use n_seeds.
no_seeds	[Deprecated] Use n_seeds.
.verbose	Boolean or integer. Controls verbosity and returns run times. FALSE -> quiet, TRUE or 1L -> normal verbosity, 2L -> detailed verbosity.

Value

SingleCellFastClusters S3 object with:

memberships data.table with cell_idx and one column per resolution (res_<value>).

stats data.table of the stability statistics, or NULL for n_seeds = 1L.

k_means_cluster Integer vector of k-means assignments, or NULL.

centroids Numeric matrix of centroids, or NULL.

resolutions Resolutions used.

with cell_indices stored as an attribute (0-indexed).

fast_mnn_gpu_sc	<i>Run fastMNN on the GPU</i>
-----------------	-------------------------------

Description

GPU counterpart of `bixverse::fast_mnn_sc()`, implementing the fast mutual nearest neighbour correction from Haghverdi, et al. Batches are merged one after the other: MNN pairs between the merged block and the next batch give correction vectors, which are smoothed with a tricube kernel and applied to every cell of that batch.

Usage

```
fast_mnn_gpu_sc(
  object,
  batch_column,
  no_embd_to_use = NULL,
  fastmnn_params = params_sc_fastmnn_gpu(),
  seed = 42L,
  .verbose = TRUE
)
```

Arguments

object	SingleCells or SingleCellsSubset class from bixverse.
batch_column	String. The column with the batch information in the obs data of the class.
no_embd_to_use	Optional integer. Number of PCs to use. If NULL all will be used.
fastmnn_params	List. Output of <code>params_sc_fastmnn_gpu()</code> .
seed	Integer. Random seed.
.verbose	Boolean or integer. Controls verbosity and returns run times. FALSE -> quiet, TRUE or 1L -> normal verbosity, 2L -> detailed verbosity.

Details

Only the neighbour searches move to the device: both MNN directions and the tricube search, three index builds per merge. Those dominate the run time on the CPU path. Centring, MNN pairing and the tricube correction are shared with the CPU implementation.

The PCA stored in the object is corrected as is. If you want the PCA on batch-aware HVGs, as `bixverse::fast_mnn_sc()` can recompute it, run `bixverse::find_hvg_batch_aware_sc()` and the PCA first.

Results match the CPU path with `knn_method = "exhaustive"` on both sides, since both searches are exact. The approximate backends will not match.

Value

The object with a "mnn_gpu" embedding added. If the batch column only has one batch, the object is returned as is with a warning.

References

Haghverdi, et al., Nat Biotechnol, 2018

find_clusters_gpu_sc *Graph-based clustering of cells on the neighbour graph (GPU)*

Description

GPU counterpart of `bixverse::find_clusters_sc()`. Runs Leiden or Louvain via `leiden_clustering_gpu()` on the neighbour graph and adds a column to the obs table. Only the level-0 local moving runs on the GPU, and only on graphs of at least 50,000 cells; smaller graphs run on the CPU. 0 is the largest cluster.

Usage

```
find_clusters_gpu_sc(
  object,
  cluster_algorithm = c("leiden", "louvain"),
  res = 1,
  name = "leiden_clustering",
  modality = c("rna", "adt", "wnn"),
  seed = 42L,
  ...
)
```

Arguments

object	SingleCells or SingleCellsSubset class from bixverse.
cluster_algorithm	String. One of c("leiden", "louvain").
res	Numeric. The modularity resolution parameter, see leiden_clustering_gpu() .
name	String. The name to add to the obs table.
modality	String. Which modality's neighbour graph to use. One of c("rna", "adt", "wnn"). The two latter options are only available for multi-modal objects.
seed	Integer. For reproducibility.
...	Further parameters for leiden_clustering_gpu() : n_iterations, beta, min_gain and local_moving.

Value

The object with the added clustering in the obs table.

find_neighbours_gpu_sc

Find GPU-accelerated neighbours for single cells

Description

This function generates kNN data using GPU-accelerated algorithms via the `bixverse.gpu` package, then turns it into a connectivities igraph for downstream clustering. See [generate_gpu_knn_sc\(\)](#) for the three searches on offer. This function lives in a separate package from the CPU-based [find_neighbours_sc\(\)](#) so that users without GPU hardware do not need to install the GPU dependencies.

Usage

```
find_neighbours_gpu_sc(
  object,
  embd_to_use = "pca",
  no_embd_to_use = NULL,
  modality = c("rna", "adt"),
  knn_method = c("nn descent", "exhaustive", "ivf"),
  nn_params = params_nn_gpu(),
  k = 15L,
  graph_params = bixverse::params_knn_graph_defaults(),
  seed = 42L,
  gpu_method = lifecycle::deprecated(),
  ivf_params = lifecycle::deprecated(),
  dist_metric = lifecycle::deprecated(),
  snn_params = lifecycle::deprecated(),
  backend = c("wgpu", "mlx"),
  .verbose = TRUE
)
```

Arguments

object	SingleCells (or SingleCellsMultiModal) class.
embd_to_use	String. The embedding to use.
no_embd_to_use	Optional integer. Number of embedding dimensions to use. If NULL all will be used.
modality	String. One of c("rna", "adt"). You can only use "adt" on SingleCellsMultiModal class.
knn_method	String. One of c("nn descent", "exhaustive", "ivf").
nn_params	List. Output of params_nn_gpu() .
k	Integer. Number of neighbours.
graph_params	List. Output of bixverse::params_knn_graph_defaults() . Controls how the kNN is turned into the connectivities graph.
seed	Integer. For reproducibility.
gpu_method	[Deprecated] Use knn_method.
ivf_params	[Deprecated] Use nn_params.
dist_metric	[Deprecated] Use params_nn_gpu(dist_metric =).
snn_params	[Deprecated] Use graph_params.
backend	String. One of c("wgpu", "mlx"). "mlx" runs the search on Apple Silicon via MLX and needs a build with the MLX backend, see mlx_available() .
.verbose	Boolean. Controls verbosity.

Value

The object with added kNN matrix and connectivities graph in the selected modality slot.

generate_gpu_knn_sc *Generate GPU kNN data for single cells*

Description

This function generates a `SingleCellNearestNeighbour` object using GPU-accelerated kNN algorithms via the `bixverse.gpu` package. Three methods are available: "exhaustive" performs an exact brute-force search on the GPU; "ivf" builds an inverted file index that partitions the embedding space into Voronoi cells and probes only a subset at query time; and "nn descent" builds a dense NNDescent graph and prunes it into a CAGRA navigational graph, which is then either beam searched or handed back as the descent left it (`params_nn_gpu(extract_knn = TRUE)`), faster, lower recall). This function is the GPU counterpart of [generate_knn_sc\(\)](#).

Usage

```
generate_gpu_knn_sc(
  object,
  embd_to_use = "pca",
  cells_to_use = NULL,
  no_embd_to_use = NULL,
  modality = c("rna", "adt"),
  knn_method = c("nn descent", "exhaustive", "ivf"),
  nn_params = params_nn_gpu(),
  k = 15L,
  seed = 42L,
  gpu_method = lifecycle::deprecated(),
  ivf_params = lifecycle::deprecated(),
  dist_metric = lifecycle::deprecated(),
  backend = c("wgpu", "mlx"),
  .verbose = TRUE
)
```

Arguments

<code>object</code>	<code>SingleCells</code> (or <code>SingleCellsMultiModal</code>) class.
<code>embd_to_use</code>	String. The embedding to use. Whichever you choose, it needs to be part of the object for the selected modality.
<code>cells_to_use</code>	Optional string vector. Cell names to include. If <code>NULL</code> all cells in the object will be used.
<code>no_embd_to_use</code>	Optional integer. Number of embedding dimensions to use. If <code>NULL</code> all will be used.
<code>modality</code>	String. One of <code>c("rna", "adt")</code> . You can only use "adt" on <code>SingleCellsMultiModal</code> class.
<code>knn_method</code>	String. One of <code>c("nn descent", "exhaustive", "ivf")</code> .
<code>nn_params</code>	List. Output of params_nn_gpu() .

k	Integer. Number of neighbours.
seed	Integer. For reproducibility.
gpu_method	[Deprecated] Use knn_method.
ivf_params	[Deprecated] Use nn_params.
dist_metric	[Deprecated] Use params_nn_gpu(dist_metric =).
backend	String. One of c("wgpu", "mlx"). "mlx" runs the search on Apple Silicon via MLX and needs a build with the MLX backend, see mlx_available() .
.verbose	Boolean or integer. Controls verbosity.

Value

Initialised `sc_knn` with the kNN data.

```
generate_knn_graph_gpu
```

Generate a k-nearest neighbour graph (GPU-accelerated)

Description

This function generates a kNN graph based on a given numeric matrix. Three different GPU-accelerated versions are available

- "exhaustive" - Exact nearest neighbour search via GPU.
- "ivf" - Inverted file index that leverages k-means clustering and probing a few of the clusters via GPU-accelerated distance calculations.
- "nndescent" - A CAGRA style nearest neighbour search on the GPU.

Usage

```
generate_knn_graph_gpu(
  data,
  k,
  knn_method = c("nndescent", "exhaustive", "ivf"),
  nn_params = params_nn_gpu(),
  seed = 42L,
  extract_knn = lifecycle::deprecated(),
  backend = c("wgpu", "mlx"),
  .verbose = TRUE
)
```

Arguments

data	Numeric matrix. The embedding or feature matrix to compute neighbours on. Rows are observations, columns are features.
k	Integer. The number of nearest neighbours to compute.
knn_method	Character. The algorithm to use for nearest neighbour search. One of c("exhaustive", "ivf", "nn descent"). Defaults to "nn descent"
nn_params	List. Output of params_nn_gpu() .
seed	Integer. For reproducibility. Defaults to 42L.
extract_knn	[Deprecated] Use the extract_knn field of params_nn_gpu() instead.
backend	String. One of c("wgpu", "mlx"). "mlx" runs the search on Apple Silicon via MLX and needs a build with the MLX backend, see mlx_available() .
.verbose	Boolean. Controls verbosity.

Value

A nearest neighbours class object with 1-indexed neighbour indices and distances. Euclidean distances are true L2, not squared.

generate_seacells_gpu_sc

Generate meta cells based on SEACells on the GPU

Description

GPU counterpart of [bixverse::generate_seacells_sc\(\)](#). Both Frank-Wolfe solves, the B-gradient argmin and the per-cell A columns, run on the WGPU backend. Kernel construction, archetype initialisation, the K^2 B bookkeeping, the RSS evaluation and the aggregation into pseudo-bulk counts stay on the CPU, so the speedup tracks how much of the run the two solves own. That share grows with n_sea_cells. Each solve falls back to its CPU sibling for that iteration if no GPU workgroup tier covers k.

Params, result class and downstream code are identical to the CPU version.

Usage

```
generate_seacells_gpu_sc(
  object,
  seacell_params = bixverse::params_sc_seacells(),
  embd_to_use = "pca",
  no_embd_to_use = NULL,
  cells_to_use = NULL,
  regenerate_knn = FALSE,
  target_size = 1e+05,
  seed = 42L,
  backend = c("wgpu", "mlx"),
  .verbose = TRUE
)
```

Arguments

object	SingleCells or SingleCellsSubset class from bixverse.
seacell_params	List. Output of <code>bixverse::params_sc_seacells()</code> . A list with the following items: • <code>n_sea_cells</code> - Number of SEA cells to detect. • <code>max_fw_iters</code> - Maximum iterations for the Frank-Wolfe algorithm per matrix update. • <code>convergence_epsilon</code> - Convergence threshold. Algorithm stops when $RSS_{change} < \epsilon * RSS(0)$. • <code>max_iter</code> - Maximum iterations to run SEACells for. • <code>min_iter</code> - Minimum iterations to run SEACells for. • <code>greedy_threshold</code> - Maximum number of cells before defaulting to rapid random selection of archetypes. • <code>graph_building</code> - Graph building method. • <code>pruning</code> - Boolean. Shall small values be pruned during the Frank- Wolfe iterations. • <code>pruning_threshold</code> - The threshold below which pruning shall be applied during Frank-Wolfe iterations. • <code>n_landmarks</code> - Optional integer. Number of landmarks for the Nystroem archetype initialisation. • <code>knn</code> - List of kNN parameters. See <code>bixverse::params_knn_defaults()</code> for available parameters and their defaults.
embd_to_use	String. The embedding to use. Atm, the only option is "pca".
no_embd_to_use	Optional integer. Number of embedding dimensions to use. If NULL all will be used.
cells_to_use	Optional string. Names of the cells to use for the generation of the SEACells. Forces a kNN rebuild on that subset.
regenerate_knn	Boolean. Shall a kNN graph be regenerated. If not, the internal one will be used.
target_size	Numeric. The library target size to normalise the meta cells to.
seed	Integer. Seed for reproducibility.
backend	String. One of <code>c("wgpu", "mlx")</code> . "mlx" runs the fit on Apple Silicon via MLX and needs a build with the MLX backend, see <code>mlx_available()</code> .
.verbose	Boolean or integer. Controls verbosity and returns run times. FALSE -> quiet, TRUE or 1L -> normal verbosity, 2L -> detailed verbosity.

Value

A `bixverse::MetaCells()` with the data generated by this meta cell aggregation method.

References

Persad, et al. Nat Biotechnol, 2023

gpu_available	<i>Is a GPU available</i>
---------------	---------------------------

Description

Reports whether a usable WGPU adapter is present. Every function in this package needs one, and without it the Rust side aborts rather than returning an error, so this is the way to branch before calling into it.

The probe initialises the same client the GPU functions use, so a TRUE means they will genuinely run rather than that a device merely exists. It is cached for the session, and the client is left warm, so the first real call afterwards skips the setup cost.

Usage

```
gpu_available()
```

Value

Boolean. TRUE when a WGPU adapter could be initialised.

Examples

```
if (gpu_available()) {
  # ... GPU path
}
```

harmony_v2_gpu_sc	<i>Run Harmony v2 (GPU)</i>
-------------------	-----------------------------

Description

A GPU-accelerated version of Harmony v2 by Patikas et al., 2026, implemented in Rust. Performs batch correction on PCA embeddings and stores the result as a "harmony_gpu" embedding in the object. Only a single batch covariate is supported on the GPU path.

Usage

```
harmony_v2_gpu_sc(
  object,
  batch_column,
  modality = c("rna", "adt"),
  harmony_params = params_sc_harmony_v2_gpu(),
  seed = 42L,
  backend = c("wgpu", "mlx"),
  .verbose = TRUE
)
```

Arguments

object	SingleCells class.
batch_column	String. Column name in the object containing the batch labels.
modality	String. One of c("rna", "adt"). You can only use "adt" on SingleCellsMultiModal class.
harmony_params	List. Output of <code>params_sc_harmony_v2_gpu()</code> .
seed	Integer. For reproducibility.
backend	String. One of c("wgpu", "mlx"). "mlx" runs on Apple Silicon via MLX and needs a build with the MLX backend, see <code>mlx_available()</code> .
.verbose	Boolean or integer. Controls verbosity and returns run times. FALSE -> quiet, TRUE or 1L -> normal verbosity, 2L -> detailed verbosity.

Value

The object with a "harmony_gpu" embedding added. If no PCA embeddings are found, returns the object unchanged with a warning.

```
install_agent_skill_gpu
```

Install the bixverse.gpu agent skill

Description

bixverse.gpu ships a skill that teaches coding agents how to *use* the package: which GPU function replaces which bixverse step, the knobs that differ, what does not run on the GPU and the traps. This copies it out of the installed package to where your agent looks for it. It complements the bixverse skill, see `bixverse::install_agent_skill()`.

The skill is versioned with the package, so re-run this after upgrading bixverse.gpu to keep the agent in sync with the code.

Usage

```
install_agent_skill_gpu(
  dest = NULL,
  agent = c("claude", "codex", "generic"),
  overwrite = FALSE
)
```

Arguments

dest	String or NULL. Directory to install into. NULL picks the default for agent: "~/claude/skills" for "claude" and "~/codex/skills" for "codex". Required for "generic". The skill lands in a bixverse-gpu subdirectory of this.
agent	String. One of c("claude", "codex", "generic"). Controls the name of the entry file and whether the frontmatter is kept.
overwrite	Boolean. Replace an existing installation. Defaults to FALSE, in which case an existing bixverse-gpu directory causes an error rather than a silent clobber.

Details

The reference files are plain markdown and work anywhere. Only the entry file differs between agents:

- "claude" writes SKILL.md with the YAML frontmatter Claude Code uses to decide when to load the skill. Discovery is automatic.
- "codex" and "generic" write AGENTS.md with the frontmatter stripped, since no other agent reads it. There is no auto-discovery, so point the agent at the directory yourself.

Value

The path the skill was written to, invisibly.

Examples

```
## Not run:
# Claude Code, into ~/.claude/skills, auto-discovered
install_agent_skill_gpu()

# Codex, into ~/.codex/skills as AGENTS.md
install_agent_skill_gpu(agent = "codex")

# refresh after a package upgrade
install_agent_skill_gpu(overwrite = TRUE)

## End(Not run)
```

k_means_cluster_gpu	<i>GPU-accelerated k-means clustering</i>
---------------------	---

Description

Performs k-means clustering as a GPU-accelerated form. In its current form uses the cubcl with wgpu backend.

Usage

```
k_means_cluster_gpu(
  data,
  k,
  kmeans_params = params_kmeans_gpu(),
  seed = 42L,
  .verbose = TRUE
)
```

Arguments

data	Numerical matrix or data frame. The data to cluster, of shape samples x features. Will be coerced to a matrix.
k	Integer. Number of clusters to create. Must be ≥ 2 .
kmeans_params	Named list. GPU-accelerated k-mean parameters, see params_kmeans_gpu() .
seed	Integer. Random seed for reproducibility. Defaults to 42L.
.verbose	Logical. Controls verbosity. Defaults to TRUE.

Value

A KMeansClusterGPU class with assignments and centroids.

leiden_clustering_gpu *GPU-accelerated Leiden or Louvain community detection on an igraph*

Description

GPU counterpart of [bixverse::leiden_clustering\(\)](#). The level-0 local moving runs on the GPU; the coarser levels, the Leiden refinement and the aggregation stay on the CPU. Graphs below 50,000 nodes, or with a node of degree above 512, run fully on the CPU with coloured local moving, as the kernel launches cost more than they save there. On one device the partition depends on seed only, but it is not identical to the CPU result: the GPU accepts moves with an order-free rule instead of the CPU's sequential re-check. Communities are numbered by size, 1 being the largest.

Usage

```
leiden_clustering_gpu(
  graph,
  resolution = 1,
  algorithm = c("leiden", "louvain"),
  beta = 0.01,
  n_iterations = NULL,
  min_gain = 0,
  local_moving = c("queue", "coloured"),
  seed = 42L
)
```

Arguments

graph	igraph. Undirected. If it has an edge attribute weight, it is used, otherwise all edges get weight 1.
resolution	Numeric. Modularity resolution parameter. Higher values give more communities.
algorithm	String. One of c("leiden", "louvain").

beta	Numeric. Randomness of the Leiden refinement merges. Ignored for Louvain.
n_iterations	Optional integer. Number of full iterations, each starting from the previous partition. A negative value iterates until the partition is stable. If NULL, 5L for Leiden and 2L for Louvain.
min_gain	Numeric. With a negative n_iterations, also stop once an iteration gains no more than this in modularity.
local_moving	String. One of c("queue", "coloured"). Applies to the coarser levels only; level 0 always runs the coloured moves on the GPU.
seed	Integer. Random seed.

Value

Integer vector with the community of each node (1-based, 1 is the largest community).

References

Traag, Waltman & van Eck, Sci Rep, 2019; Blondel, et al., J Stat Mech, 2008

load_parametric_umap *Load a parametric UMAP as a qs2 file*

Description

Load a parametric UMAP as a qs2 file

Usage

```
load_parametric_umap(path)
```

Arguments

path String. The path to the serialised trained parametric UMAP model on disk.

Value

The ParametricUmapModel.

mlx_available	<i>Is the MLX backend available</i>
---------------	-------------------------------------

Description

Reports whether this build carries the MLX backend. MLX is Apple Silicon only and opt-in at install time, since mlx-c is a system library the package cannot ship: install mlx-c (brew install mlx-c, or point MLX_C_PREFIX at an install prefix) and reinstall with BIXVERSE_GPU_MLX=true. Functions with an MLX path take backend = "mlx".

Usage

```
mlx_available()
```

Value

Boolean. TRUE when the package was built with the MLX backend.

Examples

```
if (mlx_available()) {
  # ... backend = "mlx"
}
```

nebula_gpu_sc	<i>Run NEBULA on single cells on the GPU</i>
---------------	--

Description

GPU counterpart of `bixverse::nebula_sc()`. Stage two of NEBULA, the per-gene penalised fits, runs on the WGPU backend in f32 and is finished on the host in f64. Cell ordering, gene batching, the dispersion shrinkage and the Wald test are the CPU code. Expect estimates close to the CPU ones, not identical to them.

REML is not implemented on the device, so `params_nebula_gpu()` does not carry it. Everything else, the design handling, the result class and the downstream code, is identical to the CPU version.

Usage

```
nebula_gpu_sc(
  object,
  subject_col,
  design,
  coef = NULL,
  contrast = NULL,
  genes_to_use = NULL,
```

```

    cells_to_use = NULL,
    offset = NULL,
    nebula_params = params_nebula_gpu(),
    .verbose = TRUE
)

```

Arguments

object	SingleCells or SingleCellsSubset class from bixverse.
subject_col	String. The column in the obs table holding the subject (donor) identifier. This is what the random effect is over.
design	Formula. The experimental design, evaluated against the obs table, e.g. <code>~ condition</code> or <code>~ condition + age</code> . Include the intercept.
coef	Optional integer or character. Which coefficient of the design the Wald test reports, as a 1-based column position or a column name. Defaults to the last column.
contrast	Optional numeric vector. One weight per design column. Mutually exclusive with <code>coef</code> .
genes_to_use	Optional character vector. The genes to fit. Defaults to every gene in the object, which is usually too many.
cells_to_use	Optional character vector. Names of the cells to fit, e.g. one cell type or one condition. Defaults to every cell that passed quality control. Cells that cannot be matched are dropped with a warning.
offset	Optional numeric vector. Strictly positive scaling factor per cell, aligned to the cells that survive the design. Defaults to NULL, which uses the library sizes.
nebula_params	A list, see params_nebula_gpu() . The list has the following parameters: • <code>nebula_method</code> - String. One of <code>c("ln", "h1")</code>. • <code>min_sigma</code>, <code>max_sigma</code> - Numeric. Bounds on the subject-level overdispersion. • <code>min_phi</code>, <code>max_phi</code> - Numeric. Bounds on the cell-level overdispersion. • <code>cutoff_cell</code> - Numeric. When to refit both overdispersions. • <code>kappa</code> - Numeric. When to trust the stage-one subject overdispersion. • <code>cpc</code> - Numeric. Minimum mean count per cell for a gene to be tested. • <code>mincp</code> - Integer. Minimum number of cells expressing a gene. • <code>min_subjects</code> - Integer. Minimum number of subjects whose own mean count per cell clears <code>cpc</code>. 0 switches the check off. • <code>eps</code> - Numeric. Optimiser stopping tolerance. • <code>gene_batch_size</code> - Integer. Genes read and fitted per batch. • <code>shrink_dispersion</code> - Boolean. Empirical Bayes shrinkage of the cell-level overdispersions.
.verbose	Boolean or integer. Controls verbosity and returns run times. FALSE -> quiet, TRUE or 1L -> normal verbosity, 2L -> detailed verbosity.

Value

A ScNebula class, see `bixverse::new_sc_nebula_res()`, with

- results - data.table. One row per gene that survived NEBULA's expression filter, with the Wald test and both overdispersions.
- coefficients - Numeric matrix of genes x coefficients.
- se - Numeric matrix of genes x coefficients.
- params - List. The parameters the run used.

References

He, et al., Commun Biol, 2021

nmf_gpu_sc	<i>Run single-run NMF on the GPU over single cell or meta cell data</i>
------------	---

Description

GPU counterpart of `bixverse::nmf_sc()`. Runs one HALS NMF on a chosen subset of cells and genes. The counts are uploaded once and the whole HALS loop runs on the WGPU backend; the NNDSVD initialisation stays on the CPU.

For SingleCells the counts are streamed from the Rust binary files, for MetaCells the in-memory sparse counts are used. Params, result class and downstream code are identical to the CPU version.

A single run is here for parity rather than speed. The GPU pays off when the same matrix serves many solves, so reach for `consensus_nmf_gpu_sc()` or `nmf_k_sweep_gpu_sc()` if you want the speed-up.

Usage

```
nmf_gpu_sc(
  object,
  k,
  cell_ids = NULL,
  gene_ids = NULL,
  preprocessing = "none",
  use_second_layer = TRUE,
  nmf_hals_params = bixverse::params_nmf_hals(),
  seed = 42L,
  backend = c("wgpu", "mlx"),
  .verbose = TRUE
)
```

Arguments

object	SingleCells or MetaCells class from bixverse.
k	Integer. Number of latent factors to return. At most 128, see NMF_GPU_MAX_RANK .
cell_ids	Optional character. Cell ids (or meta cell ids) to restrict the NMF to. If NULL, uses bixverse::get_cells_to_keep() for SingleCells and all meta cells for MetaCells.
gene_ids	Optional character. Gene ids to restrict the NMF to. If NULL, uses bixverse::get_hvg() on the object.
preprocessing	String. One of c("none", "sd", "sqrt_sd").
use_second_layer	Boolean. If TRUE, runs NMF on the normalised counts (recommended); if FALSE, on the raw counts.
nmf_hals_params	List, see bixverse::params_nmf_hals() .
seed	Integer. Random seed for initialisation.
backend	String. One of c("wgpu", "mlx"). "mlx" runs the HALS solves on Apple Silicon via MLX and needs a build with the MLX backend, see mlx_available() .
.verbose	Boolean or integer. Controls verbosity. FALSE -> quiet, TRUE or 1L -> normal verbosity, 2L -> detailed verbosity.

Value

An `NmfResult` object, the same class [bixverse::nmf_sc\(\)](#) returns.

nmf_k_sweep_gpu_sc	<i>Sweep k for consensus NMF on the GPU over single cell or meta cell data</i>
--------------------	--

Description

GPU counterpart of [bixverse::nmf_k_sweep_sc\(\)](#). Runs the consensus step across a range of ranks and reports stability against reconstruction error, keeping no factors. Pick the last k before stability falls away while the error curve is still coming down, then fit there with [consensus_nmf_gpu_sc\(\)](#).

Usage

```
nmf_k_sweep_gpu_sc(
  object,
  k_range,
  cell_ids = NULL,
  gene_ids = NULL,
  preprocessing = "none",
  use_second_layer = TRUE,
  nmf_hals_params = bixverse::params_nmf_hals(),
```

```

nmf_consensus_params = bixverse::params_nmf_consensus(),
n_runs = 30L,
seed = 42L,
backend = c("wgpu", "mlx"),
.verbose = TRUE
)

```

Arguments

object	SingleCells or MetaCells class from bixverse.
k_range	Integer vector. The ranks to evaluate. Every entry at least 2 and at most 128, see NMF_GPU_MAX_RANK .
cell_ids	Optional character. Cell ids (or meta cell ids) to restrict the NMF to. If NULL, uses bixverse::get_cells_to_keep() for SingleCells and all meta cells for MetaCells.
gene_ids	Optional character. Gene ids to restrict the NMF to. If NULL, uses bixverse::get_hvg() on the object.
preprocessing	String. One of c("none", "sd", "sqrt_sd").
use_second_layer	Boolean. If TRUE, runs NMF on the normalised counts (recommended); if FALSE, on the raw counts.
nmf_hals_params	List, see bixverse::params_nmf_hals() . The nmf_init field is ignored, restarts always use random initialisation.
nmf_consensus_params	List, see bixverse::params_nmf_consensus() .
n_runs	Integer. Number of restarts. At least 2.
seed	Integer. Random seed for initialisation.
backend	String. One of c("wgpu", "mlx"). "mlx" runs the HALS solves on Apple Silicon via MLX and needs a build with the MLX backend, see mlx_available() .
.verbose	Boolean or integer. Controls verbosity. FALSE -> quiet, TRUE or 1L -> normal verbosity, 2L -> detailed verbosity.

Details

This is the shape the GPU path is really for. The counts upload once and serve all $\text{length}(k_range) * n_runs$ solves, where the CPU pays full memory traffic over the matrix for every one of them. The scratch is sized once at the largest rank in k_range .

It is a diagnostic, so it leaves the object alone and hands the result back directly. `plot()` on it gives you the two curves.

Value

An `NmfKSweepResult`, which is a `data.table` with one row per k .

References

Kotliar et al., eLife, 2019

parametric_umap	<i>Parametric UMAP</i>
-----------------	------------------------

Description

Performs parametric UMAP dimensionality reduction using a neural network encoder trained on the GPU via wgpu.

Usage

```
parametric_umap(
  data,
  n_dim = 2L,
  k = 15L,
  min_dist = 0.1,
  spread = 1,
  knn_method = c("kmknn", "hns", "annoy", "nndescent", "balltree", "exhaustive"),
  nn_params = manifoldsR::params_nn(),
  parametric_umap_params = params_parametric_umap(),
  use_gpu = TRUE,
  seed = 42L,
  .verbose = TRUE
)
```

Arguments

data	Numerical matrix or data frame. The data to embed of shape samples x features. Will be coerced to a matrix.
n_dim	Integer. Number of embedding dimensions. Defaults to 2L.
k	Integer. Number of nearest neighbours. Defaults to 15L.
min_dist	Numeric. Minimum distance between embedded points. Defaults to 0.1.
spread	Numeric. Effective scale of embedded points. Defaults to 1.0.
knn_method	Character. Approximate nearest neighbour algorithm. One of "kmknn", "hns", "annoy", "nndescent", "balltree", or "exhaustive". Defaults to "kmknn".
nn_params	Named list. Nearest neighbour parameters, see params_nn() .
parametric_umap_params	Named list. Parametric UMAP parameters, see params_parametric_umap() .
use_gpu	Boolean. Shall the neural net be trained on GPU via the wgpu backend. On smaller data sets the CPU backend (flex) can be faster, since kernel launch overhead dominates.
seed	Integer. Random seed for reproducibility. Defaults to 42L.
.verbose	Boolean or integer. Controls verbosity and returns run times. FALSE -> quiet, TRUE or 1L -> normal verbosity, 2L -> detailed verbosity.

Value

A ParametricUmapModel object containing the embedding matrix and the trained encoder model.

params_kmeans_gpu	<i>Default parameters for GPU k-means</i>
-------------------	---

Description

Default parameters for GPU k-means

Usage

```
params_kmeans_gpu(
  k_means_iter = 50L,
  k_means_init = NULL,
  metric = c("euclidean", "cosine"),
  fixed = FALSE,
  quantise = FALSE
)
```

Arguments

k_means_iter	Integer. Number of k-means iterations. Defaults to 50L.
k_means_init	String or NULL. Initialisation method. One of "random", "parallel", or "plusplus". If NULL, determined on the Rust side. Defaults to NULL.
metric	String. The distance metric. One of c("euclidean", "cosine"). Defaults to "euclidean".
fixed	Boolean. Shall the algorithm be run for a fixed number of iterations, without checking for convergence. Defaults to FALSE.
quantise	Boolean. Whether to quantise data to fp16 before clustering. This can improve performance in circumstances where it is memory bound. Defaults to FALSE.

Value

A named list with the following elements:

- k_means_iter - Integer. Number of k-means iterations. Defaults to 50L.
- k_means_init - String or NULL. Initialisation method. One of "random", "parallel", or "plusplus". If NULL, determined on the Rust side. Defaults to NULL.
- metric - String. The distance metric. One of c("euclidean", "cosine"). Defaults to "euclidean".
- fixed - Boolean. Shall the algorithm be run for a fixed number of iterations, without checking for convergence. Defaults to FALSE.
- quantise - Boolean. Whether to quantise data to fp16 before clustering. This can improve performance in circumstances where it is memory bound. Defaults to FALSE.

params_knn_gpu_defaults

Default parameters for the GPU nearest neighbour backends

Description

GPU sibling of `bixverse::params_knn_defaults()`. The GPU indices take a different knob set: there is no Annoy and no HNSW on the device, so what survives is exhaustive, IVF and NN-descent.

Usage

```
params_knn_gpu_defaults()
```

Details

NN-descent builds a CAGRA graph and, with `extract_knn = TRUE`, hands that graph back rather than beam searching over it. Note that this saves the query, not the build: the descent itself dominates, and its build degree tracks `k`. NN-descent is therefore a low-`k` tool on the GPU. Above `k` of roughly 30 both exhaustive and IVF beat it, and by `k = 200` they beat it by more than an order of magnitude.

Value

A named list with the following elements:

- `k` - Integer. Number of neighbours. 0L hands the choice to Rust, which uses $\sqrt{n_cells} * 0.5$ and then adjusts for the simulated doublets. Defaults to 15L.
- `knn_method` - String. The GPU index to use. One of `c("exhaustive", "ivf", "nndescent")`. Defaults to "exhaustive".
- `ann_dist` - String. Manhattan is not supported by the GPU kernels. One of `c("euclidean", "cosine")`. Defaults to "euclidean".
- `n_list` - Integer or NULL. IVF only. Number of clusters. NULL gives $\sqrt{n}$. Defaults to NULL.
- `n_probe` - Integer or NULL. IVF only. Clusters to probe. NULL gives $\sqrt{n_list}$. Defaults to NULL.
- `graph_k` - Integer or NULL. NN-descent only. Node degree of the graph after pruning. NULL gives 30, widened to cover `k` when `extract_knn` is set. Defaults to NULL.
- `k_build` - Integer or NULL. NN-descent only. Build degree before pruning. NULL gives $\max(k, \text{floor}(1.5 * k))$. Defaults to NULL.
- `n_tree` - Integer or NULL. NN-descent only. Trees seeding the descent. Defaults to NULL.
- `delta` - Numeric. NN-descent only. Termination criterium for the descent. Defaults to 0.001.
- `rho` - Numeric or NULL. NN-descent only. Sampling rate for the descent. Defaults to NULL.
- `refine_knn` - Integer or NULL. NN-descent only. 2-hop refinement sweeps after the descent. Buys graph quality at a linear cost. NULL gives 0. Defaults to NULL.

- `beam_width` - Integer or NULL. NN-descent only. Beam width when querying. Ignored when `extract_knn` is set. Defaults to NULL.
- `max_beam_iters` - Integer or NULL. NN-descent only. Beam search iterations. Ignored when `extract_knn` is set. Defaults to NULL.
- `n_entry_points` - Integer or NULL. NN-descent only. Entry points when querying. Ignored when `extract_knn` is set. Defaults to NULL.
- `extract_knn` - Boolean. NN-descent only. Hand back the graph the descent built instead of beam searching over it. Defaults to FALSE.

params_nebula_gpu

Wrapper function for parameters for GPU NEBULA

Description

GPU counterpart to `bixverse::params_nebula()`. Same knobs and defaults, minus `rem1`: the device fit does not implement it. Stage two of NEBULA, the per-gene penalised fits, runs in `f32` on the device and is finished on the host in `f64`, so the estimates sit close to the CPU ones rather than on them.

Usage

```
params_nebula_gpu(
  nebula_method = c("ln", "hl"),
  min_sigma = 1e-04,
  min_phi = 1e-04,
  max_sigma = 10,
  max_phi = 1000,
  cutoff_cell = 20,
  kappa = 800,
  cpc = 0.005,
  mincp = 5L,
  min_subjects = 0L,
  eps = 1e-06,
  gene_batch_size = 1000L,
  shrink_dispersion = TRUE
)
```

Arguments

<code>nebula_method</code>	String. Which variant to run. NEBULA downgrades "ln" to "hl" below 30 cells per subject, as the R package does. One of <code>c("ln", "hl")</code> . Defaults to "ln".
<code>min_sigma</code>	Numeric. Lower bound on the subject-level overdispersion. Defaults to <code>1e-04</code> .
<code>min_phi</code>	Numeric. Lower bound on the cell-level overdispersion. Defaults to <code>1e-04</code> .
<code>max_sigma</code>	Numeric. Upper bound on the subject-level overdispersion. Defaults to <code>10.0</code> .

max_phi	Numeric. Upper bound on the cell-level overdispersion. Defaults to 1000.0.
cutoff_cell	Numeric. Refit both overdispersions when the product of the cells per subject and the estimated phi falls below this. Defaults to 20.0.
kappa	Numeric. Threshold on NEBULA's kappa_obs above which the subject-level overdispersion from stage one is trusted as is. Defaults to 800.0.
cpc	Numeric. Drop a gene whose mean count per cell is at most this. Defaults to 0.005.
mincp	Integer. Drop a gene expressed in fewer than this many cells. Defaults to 5L.
min_subjects	Integer. Drop a gene that fewer than this many subjects express, a subject expressing it when its own mean count per cell is above cpc. cpc and mincp pool every cell, so one subject can carry a gene through on its own. 0 switches the check off, as in the nebula package. Defaults to 0L.
eps	Numeric. Absolute stopping tolerance for the optimiser. Defaults to 1e-06.
gene_batch_size	Integer. Genes read and fitted per batch. Bounds how much of the store is resident at once and changes nothing about the answer, since NEBULA is gene-independent. Defaults to 1000L.
shrink_dispersion	Boolean. Shrink the cell-level overdispersions towards an empirical Bayes prior once the sweep is done. Defaults to TRUE.

Value

A named list with the following elements:

- nebula_method - String. Which variant to run. NEBULA downgrades "ln" to "h1" below 30 cells per subject, as the R package does. One of c("ln", "h1"). Defaults to "ln".
- min_sigma - Numeric. Lower bound on the subject-level overdispersion. Defaults to 1e-04.
- min_phi - Numeric. Lower bound on the cell-level overdispersion. Defaults to 1e-04.
- max_sigma - Numeric. Upper bound on the subject-level overdispersion. Defaults to 10.0.
- max_phi - Numeric. Upper bound on the cell-level overdispersion. Defaults to 1000.0.
- cutoff_cell - Numeric. Refit both overdispersions when the product of the cells per subject and the estimated phi falls below this. Defaults to 20.0.
- kappa - Numeric. Threshold on NEBULA's kappa_obs above which the subject-level overdispersion from stage one is trusted as is. Defaults to 800.0.
- cpc - Numeric. Drop a gene whose mean count per cell is at most this. Defaults to 0.005.
- mincp - Integer. Drop a gene expressed in fewer than this many cells. Defaults to 5L.
- min_subjects - Integer. Drop a gene that fewer than this many subjects express, a subject expressing it when its own mean count per cell is above cpc. cpc and mincp pool every cell, so one subject can carry a gene through on its own. 0 switches the check off, as in the nebula package. Defaults to 0L.
- eps - Numeric. Absolute stopping tolerance for the optimiser. Defaults to 1e-06.

- `gene_batch_size` - Integer. Genes read and fitted per batch. Bounds how much of the store is resident at once and changes nothing about the answer, since NEBULA is gene-independent. Defaults to 1000L.
- `shrink_dispersion` - Boolean. Shrink the cell-level overdispersions towards an empirical Bayes prior once the sweep is done. Defaults to TRUE.

References

He, et al., Commun Biol, 2021

params_nn_gpu

Wrapper function to generate GPU nearest neighbour parameters

Description

Wrapper function to generate GPU nearest neighbour parameters

Usage

```
params_nn_gpu(
  dist_metric = c("euclidean", "cosine"),
  n_list = NULL,
  n_probes = NULL,
  node_degree_final = NULL,
  k_build = NULL,
  n_tree = NULL,
  delta = 0.001,
  rho = NULL,
  beam_width = NULL,
  max_beam_iters = NULL,
  n_entry_points = NULL,
  extract_knn = FALSE
)
```

Arguments

<code>dist_metric</code>	String. The distance metric to use. One of <code>c("euclidean", "cosine")</code> . Defaults to "euclidean".
<code>n_list</code>	Integer or NULL. IVF GPU: Number of clusters to use. If NULL, will default to <code>sqrt(n)</code> . Defaults to NULL.
<code>n_probes</code>	Integer or NULL. IVF GPU: Number of clusters to probe. If NULL, will default to <code>sqrt(n_list)</code> . Defaults to NULL.
<code>node_degree_final</code>	Integer or NULL. Final node degree of the CAGRA navigational graph. If NULL, defaults to 30 on the Rust side. Defaults to NULL.

k_build	Integer or NULL. Number of k-neighbours during the NNDescent build phase before CAGRA pruning. If NULL, defaults to $1.5 * \text{node_degree_final}$ on the Rust side. (Cannot be smaller than <code>node_degree_final</code>) Defaults to NULL.
n_tree	Integer or NULL. CAGRA GPU: Number of trees for graph build. Automatically if NULL. Defaults to NULL.
delta	Numeric. CAGRA GPU: Early termination parameter for NN descent. Defaults to 0.001.
rho	Numeric or NULL. CAGRA GPU: Sample rate parameter for NN descent. Defaults to NULL.
beam_width	Integer or NULL. CAGRA GPU: Beam width for beam search. If not provided will be set to $\max(c(k, \text{node_degree_final}, 16L)) * 2$. Defaults to NULL.
max_beam_iters	Integer or NULL. CAGRA GPU: Maximum number of beam search iterations. If not provided, defaults to $3 * \text{beam_width}$. Defaults to NULL.
n_entry_points	Integer or NULL. CAGRA GPU: Number of entry points for beam search. If not provided, defaults to 8L. Defaults to NULL.
extract_knn	Boolean. CAGRA GPU: Skip the beam search and take the graph the NNDescent left it. Faster, slightly lower recall. Ignored by the other two searches. Defaults to FALSE.

Value

A named list with the following elements:

- `dist_metric` - String. The distance metric to use. One of `c("euclidean", "cosine")`. Defaults to "euclidean".
- `n_list` - Integer or NULL. IVF GPU: Number of clusters to use. If NULL, will default to `sqrt(n)`. Defaults to NULL.
- `n_probes` - Integer or NULL. IVF GPU: Number of clusters to probe. If NULL, will default to `sqrt(n_list)`. Defaults to NULL.
- `node_degree_final` - Integer or NULL. Final node degree of the CAGRA navigational graph. If NULL, defaults to 30 on the Rust side. Defaults to NULL.
- `k_build` - Integer or NULL. Number of k-neighbours during the NNDescent build phase before CAGRA pruning. If NULL, defaults to $1.5 * \text{node_degree_final}$ on the Rust side. (Cannot be smaller than `node_degree_final`) Defaults to NULL.
- `n_tree` - Integer or NULL. CAGRA GPU: Number of trees for graph build. Automatically if NULL. Defaults to NULL.
- `delta` - Numeric. CAGRA GPU: Early termination parameter for NN descent. Defaults to 0.001.
- `rho` - Numeric or NULL. CAGRA GPU: Sample rate parameter for NN descent. Defaults to NULL.
- `beam_width` - Integer or NULL. CAGRA GPU: Beam width for beam search. If not provided will be set to $\max(c(k, \text{node_degree_final}, 16L)) * 2$. Defaults to NULL.
- `max_beam_iters` - Integer or NULL. CAGRA GPU: Maximum number of beam search iterations. If not provided, defaults to $3 * \text{beam_width}$. Defaults to NULL.

- `n_entry_points` - Integer or NULL. CAGRA GPU: Number of entry points for beam search. If not provided, defaults to 8L. Defaults to NULL.
- `extract_knn` - Boolean. CAGRA GPU: Skip the beam search and take the graph the NNDescent left it. Faster, slightly lower recall. Ignored by the other two searches. Defaults to FALSE.

params_parametric_umap

Wrapper function to generate parametric UMAP parameters

Description

Wrapper function to generate parametric UMAP parameters

Usage

```
params_parametric_umap(
  local_connectivity = 1,
  bandwidth = 1e-05,
  mix_weight = 1,
  hidden_layers = c(128L, 64L, 32L),
  lr = 0.001,
  corr_weight = 0,
  n_epochs = 500L,
  batch_size = 256L,
  neg_sample_rate = 5L
)
```

Arguments

<code>local_connectivity</code>	Numeric. Number of nearest neighbours assumed to be at distance zero. Defaults to 1.0.
<code>bandwidth</code>	Numeric. Convergence tolerance for smooth kNN distance binary search. Defaults to 1e-05.
<code>mix_weight</code>	Numeric. Balance between fuzzy union and directed graph during symmetrisation. Defaults to 1.0.
<code>hidden_layers</code>	Integer vector. Hidden layer sizes for the MLP encoder. Defaults to c(128L, 64L, 32L).
<code>lr</code>	Numeric. Learning rate for the neural network optimiser. Defaults to 0.001.
<code>corr_weight</code>	Numeric. Coefficient for the negative Pearson correlation loss that encourages similar distances in embedding and original space. Defaults to 0.0.
<code>n_epochs</code>	Integer. Number of training epochs. Defaults to 500L.
<code>batch_size</code>	Integer. Training batch size. Defaults to 256L.
<code>neg_sample_rate</code>	Integer. Number of negative samples per positive edge. Defaults to 5L.

Value

A named list with the following elements:

- `local_connectivity` - Numeric. Number of nearest neighbours assumed to be at distance zero. Defaults to 1.0.
- `bandwidth` - Numeric. Convergence tolerance for smooth kNN distance binary search. Defaults to 1e-05.
- `mix_weight` - Numeric. Balance between fuzzy union and directed graph during symmetrisation. Defaults to 1.0.
- `hidden_layers` - Integer vector. Hidden layer sizes for the MLP encoder. Defaults to c(128L, 64L, 32L).
- `lr` - Numeric. Learning rate for the neural network optimiser. Defaults to 0.001.
- `corr_weight` - Numeric. Coefficient for the negative Pearson correlation loss that encourages similar distances in embedding and original space. Defaults to 0.0.
- `n_epochs` - Integer. Number of training epochs. Defaults to 500L.
- `batch_size` - Integer. Training batch size. Defaults to 256L.
- `neg_sample_rate` - Integer. Number of negative samples per positive edge. Defaults to 5L.

params_sc_bbknn_gpu	<i>Wrapper function for the GPU BBKNN parameters</i>
---------------------	--

Description

GPU counterpart to `bixverse::params_sc_bbknn()`. Same BBKNN knobs, but the kNN block is the GPU one, see `params_knn_gpu_defaults()`.

Usage

```
params_sc_bbknn_gpu(
  neighbours_within_batch = 3L,
  set_op_mix_ratio = 1,
  local_connectivity = 1,
  trim = NULL,
  knn = list()
)
```

Arguments

`neighbours_within_batch`
Integer. Number of neighbours to consider per batch. Defaults to 3L.

`set_op_mix_ratio`
Numeric. Mixing ratio between union (1.0) and intersection (0.0). Defaults to 1.0.

local_connectivity	Numeric. UMAP connectivity computation parameter, how many nearest neighbours of each cell are assumed to be fully connected. Defaults to 1.0.
trim	Integer or NULL. Trim the neighbours of each cell to these many top connectivities. May help with population independence and improve the tidiness of clustering. If NULL, it defaults to 10 * neighbours_within_batch. Defaults to NULL.
knn	List. Optional overrides for the kNN block. See params_knn_gpu_defaults() for the available elements. Without k, extract_knn. Unknown elements are an error. Defaults to list().

Details

Two keys of the GPU kNN block do nothing here and are rejected rather than silently ignored. k is set by neighbours_within_batch, and extract_knn only applies to a self-query, whereas BBKNN builds one index per batch and queries each with every cell.

Value

A named list with the following elements:

- neighbours_within_batch - Integer. Number of neighbours to consider per batch. Defaults to 3L.
- set_op_mix_ratio - Numeric. Mixing ratio between union (1.0) and intersection (0.0). Defaults to 1.0.
- local_connectivity - Numeric. UMAP connectivity computation parameter, how many nearest neighbours of each cell are assumed to be fully connected. Defaults to 1.0.
- trim - Integer or NULL. Trim the neighbours of each cell to these many top connectivities. May help with population independence and improve the tidiness of clustering. If NULL, it defaults to 10 * neighbours_within_batch. Defaults to NULL.
- The elements of [params_knn_gpu_defaults\(\)](#), overridden by knn, spliced in at this position.

References

Polański, et al., Bioinformatics, 2020

params_sc_fast_cluster_gpu

Default parameters for GPU fast clustering

Description

GPU counterpart to [bixverse::params_sc_fast_cluster\(\)](#). Fast clustering coarsens the cells with k-means, builds a graph over the centroids and runs Louvain or Leiden on it. The community of each centroid is handed back to its cells. Only the k-means (and the cell-mediated top-m search) run on the GPU. The mini-batch k-means knobs are gone: the GPU k-means is full-batch Lloyd's.

Usage

```

params_sc_fast_cluster_gpu(
  n_centroids = NULL,
  k_means_iter = 50L,
  k_means_init = NULL,
  fixed = TRUE,
  quantise = FALSE,
  dist = c("euclidean", "cosine"),
  cell_mediated = FALSE,
  n_anchors = 5L,
  centroid_k = 15L,
  centroid_graph = list(),
  algorithm = c("louvain", "leiden"),
  n_iterations = NULL,
  beta = 0.01,
  min_gain = 0,
  local_moving = c("queue", "coloured"),
  multi_level = TRUE,
  max_evals = 10L
)

```

Arguments

n_centroids	Integer or NULL. Number of k-means centroids. If NULL, defaults to <code>floor(sqrt(n_cells))</code> . Defaults to NULL.
k_means_iter	Integer. Maximum number of k-means iterations. Defaults to 50L.
k_means_init	String or NULL. Initialisation method. One of "random", "parallel" or "plusplus". If NULL, picked on the Rust side based on the number of centroids. Defaults to NULL.
fixed	Boolean. Shall k-means run for a fixed number of iterations, without checking for convergence. wgpu only; MLX always runs every iteration. Defaults to TRUE.
quantise	Boolean. Shall the data buffer be held at fp16 on the GPU. Halves the buffer and helps when the assignment kernels are memory bound. wgpu only. Defaults to FALSE.
dist	String. Distance for the k-means and the centroid graph. One of <code>c("euclidean", "cosine")</code> . Defaults to "euclidean".
cell_mediated	Boolean. Shall the cell-mediated centroid graph from Secuer be built instead of the kNN graph on the centroids. Centroids are then linked by the cells that sit near both, so two centroids that are close in space but have no cells between them stay apart. If TRUE, centroid_k and centroid_graph are ignored. Defaults to FALSE.
n_anchors	Integer. Number of nearest centroids each cell links to in the cell-mediated graph. Only used if cell_mediated = TRUE. Defaults to 5L.
centroid_k	Integer. Neighbours per centroid in the exact centroid kNN. Only used if cell_mediated = FALSE. Defaults to 15L.

centroid_graph	List. Optional overrides for how the centroid kNN is turned into a graph. See bixverse::params_knn_graph_defaults() . Only used if cell_mediated = FALSE. Defaults to list().
algorithm	String. Community detection on the centroid graph. One of c("louvain", "leiden"). Defaults to "louvain".
n_iterations	Integer or NULL. Number of iterations of the community detection. A negative value iterates until stable. If NULL, the bixverse default for the chosen algorithm. Defaults to NULL.
beta	Numeric. Leiden only. Randomness of the refinement step. Defaults to 0.01.
min_gain	Numeric. With a negative n_iterations, also stop once an iteration gains no more than this in modularity. Defaults to 0.0.
local_moving	String. "queue" is Traag's sequential queue. "coloured" moves the nodes of one colour class in parallel, which only kicks in on larger levels of the graph. One of c("queue", "coloured"). Defaults to "queue".
multi_level	Boolean. Shall the communities be aggregated and refined in levels. Defaults to TRUE.
max_evals	Integer. Cap on the node evaluations per local moving phase, in units of the number of nodes. Defaults to 10L.

Value

A named list with the following elements:

- n_centroids - Integer or NULL. Number of k-means centroids. If NULL, defaults to floor(sqrt(n_cells)). Defaults to NULL.
- k_means_iter - Integer. Maximum number of k-means iterations. Defaults to 50L.
- k_means_init - String or NULL. Initialisation method. One of "random", "parallel" or "plusplus". If NULL, picked on the Rust side based on the number of centroids. Defaults to NULL.
- fixed - Boolean. Shall k-means run for a fixed number of iterations, without checking for convergence. wgpu only; MLX always runs every iteration. Defaults to TRUE.
- quantise - Boolean. Shall the data buffer be held at fp16 on the GPU. Halves the buffer and helps when the assignment kernels are memory bound. wgpu only. Defaults to FALSE.
- dist - String. Distance for the k-means and the centroid graph. One of c("euclidean", "cosine"). Defaults to "euclidean".
- cell_mediated - Boolean. Shall the cell-mediated centroid graph from Secuer be built instead of the kNN graph on the centroids. Centroids are then linked by the cells that sit near both, so two centroids that are close in space but have no cells between them stay apart. If TRUE, centroid_k and centroid_graph are ignored. Defaults to FALSE.
- n_anchors - Integer. Number of nearest centroids each cell links to in the cell-mediated graph. Only used if cell_mediated = TRUE. Defaults to 5L.
- centroid_k - Integer. Neighbours per centroid in the exact centroid kNN. Only used if cell_mediated = FALSE. Defaults to 15L.
- The elements of the base list, overridden by centroid_graph, spliced in at this position.

- `algorithm` - String. Community detection on the centroid graph. One of `c("louvain", "leiden")`. Defaults to `"louvain"`.
- `n_iterations` - Integer or NULL. Number of iterations of the community detection. A negative value iterates until stable. If NULL, the bixverse default for the chosen algorithm. Defaults to NULL.
- `beta` - Numeric. Leiden only. Randomness of the refinement step. Defaults to `0.01`.
- `min_gain` - Numeric. With a negative `n_iterations`, also stop once an iteration gains no more than this in modularity. Defaults to `0.0`.
- `local_moving` - String. `"queue"` is Traag's sequential queue. `"coloured"` moves the nodes of one colour class in parallel, which only kicks in on larger levels of the graph. One of `c("queue", "coloured")`. Defaults to `"queue"`.
- `multi_level` - Boolean. Shall the communities be aggregated and refined in levels. Defaults to TRUE.
- `max_evals` - Integer. Cap on the node evaluations per local moving phase, in units of the number of nodes. Defaults to `10L`.

References

Wei, et al., PLoS Comput Biol, 2022.

`params_sc_fastmnn_gpu` *Wrapper function for the GPU fastMNN parameters*

Description

GPU counterpart to `bixverse::params_sc_fastmnn()`. Same fastMNN knobs minus the PCA ones, since the GPU path corrects the PCA already stored in the object. The kNN block is the GPU one, see `params_knn_gpu_defaults()`, with `k` and `ann_dist` defaulting to the CPU values.

Usage

```
params_sc_fastmnn_gpu(ndist = 3, cos_norm = TRUE, knn = list())
```

Arguments

<code>ndist</code>	Numeric. Number of median distances for the tricube kernel bandwidth. Defaults to <code>3.0</code> .
<code>cos_norm</code>	Boolean. Apply cosine normalisation before computing distances. Defaults to TRUE.
<code>knn</code>	List. Optional overrides for the kNN block. See <code>params_knn_gpu_defaults()</code> for the available elements. Without <code>extract_knn</code> . Unknown elements are an error. Defaults to <code>list()</code> .

Details

extract_knn is rejected rather than silently ignored: every search in fastMNN is a cross-query between two sets of cells, and extraction only applies to a self-query. $k = 0$ is rejected as well, since fastMNN has no data-driven fallback for it.

Value

A named list with the following elements:

- ndist - Numeric. Number of median distances for the tricube kernel bandwidth. Defaults to 3.0.
- cos_norm - Boolean. Apply cosine normalisation before computing distances. Defaults to TRUE.
- The elements of `params_knn_gpu_defaults()`, overridden by knn, spliced in at this position.

References

Haghverdi, et al., Nat Biotechnol, 2018

params_sc_harmony_v2_gpu

Default parameters for Harmony v2 GPU batch correction

Description

Default parameters for Harmony v2 GPU batch correction

Usage

```
params_sc_harmony_v2_gpu(
  k = NULL,
  sigma = 0.1,
  theta = 2,
  lambda = 1,
  max_iter_kmeans = 4L,
  max_iter_harmony = 10L,
  epsilon_kmeans = 0.001,
  epsilon_harmony = 0.01,
  window_size = 3L,
  alpha = 0.2,
  tau = 0,
  batch_proportion_cutoff = 1e-05,
  use_dynamic_lambda = FALSE,
  csr_cube_count = 256L,
  k_means_iter = 10L,
  k_means_init = NULL,
```

```

    fixed = FALSE,
    quantise = FALSE
)

```

Arguments

k	Integer or NULL. Number of clusters for k-means clustering. If not provided, it will be automatically determined as $\min(\text{round}(N / 30), 100)$. Defaults to NULL.
sigma	Numeric vector. Per-cluster diversity weights. Either a single value (broadcast to all clusters) or a vector of length k. Defaults to 0.1.
theta	Numeric vector. Per-variable diversity penalty. Must be a single value; only one batch covariate is supported on the GPU path. Defaults to 2.0.
lambda	Numeric vector. Ridge regression penalty for the linear model. Typically a single value. Ignored when <code>use_dynamic_lambda = TRUE</code> . Defaults to 1.0.
max_iter_kmeans	Integer. Maximum number of k-means Jacobi sweeps per Harmony round. Defaults to 4L.
max_iter_harmony	Integer. Maximum number of Harmony outer iterations. Defaults to 10L.
epsilon_kmeans	Numeric. Convergence threshold for k-means clustering. Defaults to 0.001.
epsilon_harmony	Numeric. Convergence threshold for Harmony. Defaults to 0.01.
window_size	Integer. Number of previous iterations to consider when checking convergence. Defaults to 3L.
alpha	Numeric. Scaling factor for dynamic lambda estimation. Must be in (0, 1). Only relevant when <code>use_dynamic_lambda = TRUE</code> . Defaults to 0.2.
tau	Numeric. Scaling factor for theta based on batch size. A value of 0 disables batch-size scaling of theta. Defaults to 0.0.
batch_proportion_cutoff	Numeric. Cutoff for pruning batches with small proportions during ridge regression. Defaults to $1e-05$.
use_dynamic_lambda	Boolean. If TRUE, lambda is estimated dynamically per cluster instead of using the fixed lambda value. Defaults to FALSE.
csr_cube_count	Integer. Number of parallel thread groups used when building the level-CSR index on the GPU. Adjust for your hardware if needed. Defaults to 256L.
k_means_iter	Integer. Maximum number of k-means iterations for the initial centroid computation. Defaults to 10L.
k_means_init	String or NULL. Initialisation strategy for k-means. Defaults to NULL.
fixed	Boolean. If TRUE, centroids are fixed after initialisation. Defaults to FALSE.
quantise	Boolean. If TRUE, quantises intermediate values to f16 during k-means. Defaults to FALSE.

Value

A named list with the following elements:

- `k` - Integer or NULL. Number of clusters for k-means clustering. If not provided, it will be automatically determined as $\min(\text{round}(N / 30), 100)$. Defaults to NULL.
- `sigma` - Numeric vector. Per-cluster diversity weights. Either a single value (broadcast to all clusters) or a vector of length `k`. Defaults to `0.1`.
- `theta` - Numeric vector. Per-variable diversity penalty. Must be a single value; only one batch covariate is supported on the GPU path. Defaults to `2.0`.
- `lambda` - Numeric vector. Ridge regression penalty for the linear model. Typically a single value. Ignored when `use_dynamic_lambda = TRUE`. Defaults to `1.0`.
- `max_iter_kmeans` - Integer. Maximum number of k-means Jacobi sweeps per Harmony round. Defaults to `4L`.
- `max_iter_harmony` - Integer. Maximum number of Harmony outer iterations. Defaults to `10L`.
- `epsilon_kmeans` - Numeric. Convergence threshold for k-means clustering. Defaults to `0.001`.
- `epsilon_harmony` - Numeric. Convergence threshold for Harmony. Defaults to `0.01`.
- `window_size` - Integer. Number of previous iterations to consider when checking convergence. Defaults to `3L`.
- `alpha` - Numeric. Scaling factor for dynamic lambda estimation. Must be in $(0, 1)$. Only relevant when `use_dynamic_lambda = TRUE`. Defaults to `0.2`.
- `tau` - Numeric. Scaling factor for theta based on batch size. A value of 0 disables batch-size scaling of theta. Defaults to `0.0`.
- `batch_proportion_cutoff` - Numeric. Cutoff for pruning batches with small proportions during ridge regression. Defaults to `1e-05`.
- `use_dynamic_lambda` - Boolean. If TRUE, lambda is estimated dynamically per cluster instead of using the fixed lambda value. Defaults to FALSE.
- `csr_cube_count` - Integer. Number of parallel thread groups used when building the level-CSR index on the GPU. Adjust for your hardware if needed. Defaults to `256L`.
- `k_means_iter` - Integer. Maximum number of k-means iterations for the initial centroid computation. Defaults to `10L`.
- `k_means_init` - String or NULL. Initialisation strategy for k-means. Defaults to NULL.
- `fixed` - Boolean. If TRUE, centroids are fixed after initialisation. Defaults to FALSE.
- `quantise` - Boolean. If TRUE, quantises intermediate values to f16 during k-means. Defaults to FALSE.

params_scrublet_gpu *Wrapper function for GPU Scrublet doublet detection parameters*

Description

GPU counterpart to `bixverse::params_scrublet()`. Two differences from the CPU list. The `pca` sub-list is gone: the GPU SVD is always randomised, so `svd_solver` has nothing to switch and `no_pcs` is a plain argument. And the `knn` block is backend dependent, see `knn_backend`.

Usage

```
params_scrublet_gpu(
  sim_doublet_ratio = 1.5,
  expected_doublet_rate = 0.1,
  stdev_doublet_rate = 0.02,
  n_bins_histogram = 100L,
  manual_threshold = NULL,
  no_pcs = 30L,
  normalisation = list(),
  hvg = list(),
  knn_backend = c("gpu", "cpu"),
  knn = list(k = 0L)
)
```

Arguments

<code>sim_doublet_ratio</code>	Numeric. Number of doublets to simulate relative to the number of observed cells. Defaults to 1.5.
<code>expected_doublet_rate</code>	Numeric in $[0, 1]$. Expected doublet rate, typically 0.05-0.10 depending on cell loading. Defaults to 0.1.
<code>stdev_doublet_rate</code>	Numeric in $[0, 1]$. Uncertainty in the expected doublet rate. Defaults to 0.02.
<code>n_bins_histogram</code>	Integer. Histogram bins for the Otsu threshold search. Defaults to 100L.
<code>manual_threshold</code>	Optional numeric. Fixed doublet score threshold. If NULL, Otsu's method picks it.
<code>no_pcs</code>	Integer. Number of principal components. Defaults to 30L.
<code>normalisation</code>	List. Optional overrides. See <code>bixverse::params_norm_doublets_defaults()</code> for the available parameters: <code>log_transform</code> , <code>mean_center</code> , <code>normalise_variance</code> , <code>target_size</code> .
<code>hvg</code>	List. Optional overrides. See <code>bixverse::params_hvg_defaults()</code> for the available parameters: <code>min_gene_var_pct1</code> , <code>hvg_method</code> , <code>loess_span</code> , <code>clip_max</code> , <code>n_bins</code> , <code>binning_strategy</code> .

knn_backend	String. One of "gpu" or "cpu". Picks which nearest neighbour index runs over the combined observed-plus-simulated embedding, and with it which keys knn accepts. "gpu" is the fast default; "cpu" buys the exact CPU indices at the cost of a host round trip on a matrix that is $(1 + \text{sim_doublet_ratio}) * \text{n_cells}$ rows tall.
knn	List. Optional overrides for the kNN block. Validated against <code>params_knn_gpu_defaults()</code> when <code>knn_backend = "gpu"</code> and against <code>bixverse::params_knn_defaults()</code> when <code>knn_backend = "cpu"</code> . Unknown keys are an error, not a silent pass-through. Defaults to <code>list(k = 0L)</code> , which asks Rust to pick k.

Details

Leave `knn_method` alone unless you have a reason. "exhaustive" is the default and is the right answer for Scrublet on the GPU arm.

Scrublet queries at a high k by construction: the count is taken over an embedding $(1 + \text{sim_doublet_ratio}) * \text{n_cells}$ rows tall, and `k = 0L` then scales k by the same factor, so a 20k-cell run searches at k around 175. Exhaustive barely notices k, since the scan is the cost and k only sizes the top-k selection. NN-descent notices a lot: its build degree tracks k, so the descent does more work per node as k climbs. "ivf" is the one worth trying: it was the quickest of the three across that sweep and holds a Pearson above 0.99 against exhaustive.

Recall matters more here than elsewhere. The doublet score is a neighbour count, so a backend that drops neighbours biases every score downwards.

Value

A flat named list with all GPU Scrublet parameters.

References

Wolock, et al., Cell Syst, 2020

params_tsne_gpu	<i>Wrapper function to generate t-SNE parameters (GPU version)</i>
-----------------	--

Description

Wrapper function to generate t-SNE parameters (GPU version)

Usage

```
params_tsne_gpu(
  lr = NULL,
  n_epochs = 1000L,
  early_exag_iter = 250L,
  early_exag_factor = 12,
  late_exag_factor = NULL,
  theta = 0.5,
```

```

    n_interp_points = 3L,
    max_depth = 7L,
    init = c("pca", "spectral", "random"),
    randomised = TRUE
)

```

Arguments

lr	Numeric or NULL. Learning rate. If NULL (the default), the Rust backend sets it to $\max((n_samples / 12), 200)$, following the N-dependent heuristic of Belkina et al. (2019). Defaults to NULL.
n_epochs	Integer. Number of optimisation epochs. Defaults to 1000L.
early_exag_iter	Integer. Number of early exaggeration iterations. Defaults to 250L.
early_exag_factor	Numeric. Early exaggeration factor. Defaults to 12.0.
late_exag_factor	Numeric or NULL. If you wish to also use late exaggerations. Can be useful on large data sets (set it to 2.0 to 4.0). Defaults to NULL.
theta	Numeric. Barnes-Hut approximation angle. Lower values increase accuracy at the cost of speed. Defaults to 0.5.
n_interp_points	Integer. Number of interpolation points per grid cell for FFT acceleration ("fft", "fft_3k" and "fft_3k_gpu"). Defaults to 3L.
max_depth	Integer. Maximum Barnes-Hut tree depth for the quick-and-dirty optimiser ("bh_qd"). Smaller is faster and coarser; qdtsne recommends 7 to 10. No effect on the other optimisers. Defaults to 7L.
init	String. Embedding initialisation method. One of c("pca", "spectral", "random"). Defaults to "pca".
randomised	Boolean. Use randomised SVD for PCA initialisation. Defaults to TRUE.

Value

A named list with the following elements:

- lr - Numeric or NULL. Learning rate. If NULL (the default), the Rust backend sets it to $\max((n_samples / 12), 200)$, following the N-dependent heuristic of Belkina et al. (2019). Defaults to NULL.
- n_epochs - Integer. Number of optimisation epochs. Defaults to 1000L.
- early_exag_iter - Integer. Number of early exaggeration iterations. Defaults to 250L.
- early_exag_factor - Numeric. Early exaggeration factor. Defaults to 12.0.
- late_exag_factor - Numeric or NULL. If you wish to also use late exaggerations. Can be useful on large data sets (set it to 2.0 to 4.0). Defaults to NULL.
- theta - Numeric. Barnes-Hut approximation angle. Lower values increase accuracy at the cost of speed. Defaults to 0.5.

- `n_interp_points` - Integer. Number of interpolation points per grid cell for FFT acceleration ("`fft`", "`fft_3k`" and "`fft_3k_gpu`"). Defaults to 3L.
- `max_depth` - Integer. Maximum Barnes-Hut tree depth for the quick-and-dirty optimiser ("`bh_qd`"). Smaller is faster and coarser; qdtsne recommends 7 to 10. No effect on the other optimisers. Defaults to 7L.
- `init` - String. Embedding initialisation method. One of `c("pca", "spectral", "random")`. Defaults to "`pca`".
- `randomised` - Boolean. Use randomised SVD for PCA initialisation. Defaults to `TRUE`.

References

Belkina, et al., Nat. Commun., 2019

<code>params_umap_gpu</code>	<i>Wrapper function to generate UMAP parameters (GPU version)</i>
------------------------------	---

Description

Wrapper function to generate UMAP parameters (GPU version)

Usage

```
params_umap_gpu(
  local_connectivity = 1,
  bandwidth = 1e-05,
  mix_weight = 1,
  lr = 1,
  n_epochs = NULL,
  neg_sample_rate = 5L,
  gamma = 1,
  optimiser = c("adam_gpu", "adam_parallel", "sgd", "adam"),
  init = c("spectral", "pca", "random"),
  randomised = FALSE
)
```

Arguments

<code>local_connectivity</code>	Numeric. Number of nearest neighbours assumed to be at distance zero. Defaults to 1.0.
<code>bandwidth</code>	Numeric. Convergence tolerance for smooth kNN distance binary search. Defaults to 1e-05.
<code>mix_weight</code>	Numeric. Balance between fuzzy union and directed graph during symmetrisation. Defaults to 1.0.
<code>lr</code>	Numeric. Learning rate. Defaults to 1.0.

n_epochs	Integer or NULL. Number of optimisation epochs. Defaults to NULL.
neg_sample_rate	Integer. Number of negative samples per positive sample. Defaults to 5L.
gamma	Numeric. Repulsion strength. Defaults to 1.0.
optimiser	String. The optimiser for the embedding. One of c("adam_gpu", "adam_parallel", "sgd", "adam"). Defaults to "adam_gpu".
init	String. Embedding initialisation method. One of c("spectral", "pca", "random"). Defaults to "spectral".
randomised	Boolean. Use randomised SVD for PCA initialisation. Defaults to FALSE.

Value

A named list with the following elements:

- local_connectivity - Numeric. Number of nearest neighbours assumed to be at distance zero. Defaults to 1.0.
- bandwidth - Numeric. Convergence tolerance for smooth kNN distance binary search. Defaults to 1e-05.
- mix_weight - Numeric. Balance between fuzzy union and directed graph during symmetrisation. Defaults to 1.0.
- lr - Numeric. Learning rate. Defaults to 1.0.
- n_epochs - Integer or NULL. Number of optimisation epochs. Defaults to NULL.
- neg_sample_rate - Integer. Number of negative samples per positive sample. Defaults to 5L.
- gamma - Numeric. Repulsion strength. Defaults to 1.0.
- optimiser - String. The optimiser for the embedding. One of c("adam_gpu", "adam_parallel", "sgd", "adam"). Defaults to "adam_gpu".
- init - String. Embedding initialisation method. One of c("spectral", "pca", "random"). Defaults to "spectral".
- randomised - Boolean. Use randomised SVD for PCA initialisation. Defaults to FALSE.

```
predict.ParametricUmapModel
```

Predict embeddings for new data using a trained parametric UMAP model

Description

Predict embeddings for new data using a trained parametric UMAP model

Usage

```
## S3 method for class 'ParametricUmapModel'
predict(object, newdata, ...)
```

Arguments

object	A ParametricUmapModel object.
newdata	Numerical matrix or data frame. New data of shape samples x features. Must have the same number of features as the training data.
...	Ignored.

Value

A numerical matrix with dimensions samples x n_dim.

rs_cor_gpu	<i>GPU-accelerated correlation calculations</i>
------------	---

Description

[Experimental] GPU-accelerated pairwise column correlations. Has the options of Pearson and Spearman correlation coefficient calculations.

Usage

rs_cor_gpu(x, spearman, backend = "wgpu", verbose)

Arguments

x	Numerical matrix. The matrix for which to calculate the column pairwise correlation matrix.
spearman	Boolean. Shall the Spearman correlation be calculated instead of Pearson.
backend	String. "wgpu" or "mlx" (Apple Silicon, needs a build with the MLX backend, see mlx_available()).
verbose	Boolean. Controls verbosity of the function.

Value

The correlation matrix

rs_cov_gpu	<i>GPU-accelerated covariance calculations</i>
------------	--

Description

[Experimental] GPU-accelerated pairwise column co-variance calculation.

Usage

```
rs_cov_gpu(x, backend = "wgpu", verbose)
```

Arguments

x	Numerical matrix. The matrix for which to calculate the column pairwise co-variance matrix.
backend	String. "wgpu" or "mlx" (Apple Silicon, needs a build with the MLX backend, see mlx_available()).
verbose	Boolean. Controls verbosity of the function.

Value

The covariance matrix

rs_fast_mnn_gpu	<i>GPU: fastMNN batch correction</i>
-----------------	--------------------------------------

Description

[Experimental] GPU equivalent of `bixverse::rs_mnn`, implementing the fast mutual nearest neighbour correction from Haghverdi, et al. The MNN searches and the tricube neighbour search run on the WGPU backend; everything else is shared with the CPU implementation.

Usage

```
rs_fast_mnn_gpu(embd, batch_labels, fastmnn_params, seed, verbose)
```

Arguments

embd	Numerical matrix. The embedding to correct, usually PCA. Rows represent cells.
batch_labels	Integer vector. These represent to which batch a given cell belongs. Needs to be 0-indexed!
fastmnn_params	List. Parameter list, see params_sc_fastmnn_gpu() .
seed	Integer. Seed for reproducibility purposes.
verbose	Integer. 0L - quiet; 1L - normal verbosity; 2L - detailed verbosity.

Value

The batch-corrected embedding, cells x dimensions, in the input cell order.

rs_gpu_available	<i>Check whether a usable GPU adapter is present</i>
------------------	--

Description

Probes for a WGPU adapter by initialising the same client every GPU function in this package goes through, so a TRUE here means those functions will actually run rather than that a device merely exists. The result is cached for the session and the client stays warm, so the first real call after a successful probe skips the setup cost.

Usage

```
rs_gpu_available()
```

Value

Boolean. TRUE when a WGPU adapter could be initialised.

rs_gpu_knn	<i>Generate a GPU-accelerated kNN graph</i>
------------	---

Description

[Experimental] Builds a kNN graph from an embedding matrix on the wgpu backend. Three searches are available: an exact brute-force scan, an IVF index that probes a subset of Voronoi cells, and a CAGRA-style NNDescent graph that is either beam searched or handed back as the descent left it.

Euclidean distances come back as true L2, not squared.

Usage

```
rs_gpu_knn(embd, k, knn_method, nn_params, seed, backend = "wgpu", verbose)
```

Arguments

embd	Numeric matrix of embeddings, cells x features.
k	Integer. Number of neighbours to return, self excluded.
knn_method	String. One of c("nn descent", "exhaustive", "ivf").
nn_params	A named list with the parameters, see params_nn_gpu()
seed	Integer. Random seed for reproducibility.
backend	String. "wgpu" or "mlx" (Apple Silicon, needs a build with the MLX backend, see mlx_available()).
verbose	Integer. 0L - quiet; 1L - normal verbosity; 2L - detailed verbosity.

Value

A named list with:

- `indices` - Integer matrix of shape `cells x k` with 0-based neighbour indices.
- `dist` - Numeric matrix of shape `cells x k` with distances to the neighbours.
- `dist_metric` - Character. The distance metric used.

`rs_gpu_plane_ops`
Check whether the GPU adapter runs plane (subgroup) operations

Description

Advertising plane operations is not enough: the paravirtualised GPU of a macOS VM, as on the `macos-15-intel` GitHub runners, reports planes of 4 to 64 lanes but silently drops every dispatch that uses one. This launches a single plane reduction and checks the answer. The result is cached for the session.

Usage

```
rs_gpu_plane_ops()
```

Value

Boolean. TRUE when a GPU adapter is present and a plane reduction on it returns the plane width.

`rs_harmony_v2_gpu`
Harmony batch correction in Rust (version 2, GPU-accelerated)

Description

[Experimental] This function implements the GPU-accelerated version 2 Harmony algorithm from Patikas, et al., 2026. Only a single batch covariate is supported.

Usage

```
rs_harmony_v2_gpu(
  pca,
  harmony_params,
  batch_labels,
  seed,
  backend = "wgpu",
  verbose
)
```

Arguments

pca	Numerical matrix, i.e., the PCA matrix you want to correct.
harmony_params	List. The parameters for the Harmony (v2) GPU algorithm.
batch_labels	List. Must contain exactly one element: a 0-indexed integer vector representing the batch effects you wish to regress out.
seed	Integer. Seed for reproducibility purposes.
backend	String. "wgpu" or "mlx" (Apple Silicon, needs a build with the MLX backend, see mlx_available()).
verbose	Integer. 0L - quiet; 1L - normal verbosity; 2L - detailed verbosity.

Value

The batch-corrected Harmony (v2) embedding space.

rs_kmeans_gpu	<i>GPU-accelerated k-means</i>
---------------	--------------------------------

Description

[Experimental] A GPU-accelerated k-means version leveraging the wgpu backend via cubcl.

Usage

```
rs_kmeans_gpu(data, dist, n_centroids, kmeans_params, seed, verbose)
```

Arguments

data	Numeric matrix. Samples x features.
dist	String. Distance metric to use. One of c("euclidean", "cosine").
n_centroids	Integer. Number of clusters, centroids to identify.
kmeans_params	Named list. Contains specific parameters for the GPU- accelerated k-means.
seed	Integer. Seed for reproducibility.
verbose	Boolean. Controls verbosity of the function.

Value

A list with

- centroids - The centroids matrix (centroids x features)
- assignments - The cluster assignments of the data. (1-indexed.)

rs_leiden_clustering_gpu

GPU-accelerated Leiden and Louvain clustering

Description

[Experimental] GPU twin of `bixverse::rs_leiden_clustering`. Level-0 local moving runs on the `wgpu` backend; coarser levels, refinement and aggregation stay on the CPU. Graphs below 50,000 nodes, or with a node of degree above 512, run fully on the CPU with coloured local moving. On one device the result depends on the seed only.

Usage

```
rs_leiden_clustering_gpu(from, to, weights, n_nodes, leiden_params)
```

Arguments

<code>from</code>	Integer vector. 1-based source node of each edge.
<code>to</code>	Integer vector. 1-based target node of each edge. Same length as <code>from</code> .
<code>weights</code>	Numeric vector. Edge weights, same length as <code>from</code> .
<code>n_nodes</code>	Integer. Number of nodes in the graph.
<code>leiden_params</code>	List. Named list with the optional fields <code>resolution</code> , <code>beta</code> , <code>n_iterations</code> (negative: until stable), <code>min_gain</code> , <code>refine</code> (TRUE for Leiden, FALSE for Louvain), <code>local_moving</code> (coarser levels only) and <code>seed</code> .

Value

Integer vector with the 1-based community of each node; 1 is the largest community.

rs_parametric_umap

Parametric UMAP implementation

Description

[Experimental] Trains a neural network encoder to learn a mapping from the input space to a low-dimensional embedding that preserves the UMAP graph structure. Supports both GPU (`wgpu`) and CPU (`burn flex CPU`) backends. For small to medium data sets (fewer than ~10k samples or narrow hidden layers), the CPU backend is typically faster owing to GPU kernel dispatch overhead.

Usage

```
rs_parametric_umap(
  data,
  n_dim,
  k,
  min_dist,
  spread,
  parametric_params,
  seed,
  verbose,
  use_gpu
)
```

Arguments

data	Numerical matrix. Data of dimensions samples x features.
n_dim	Integer. Number of embedding dimensions.
k	Integer. Number of nearest neighbours for graph construction.
min_dist	Numeric. Minimum distance between embedded points.
spread	Numeric. Effective scale of embedded points.
parametric_params	Named list. Merged parametric UMAP parameters containing nearest neighbour, graph, and training configuration.
seed	Integer. Seed for reproducibility.
verbose	Integer. 0L - quiet; 1L - normal verbosity; 2L - detailed verbosity.
use_gpu	Logical. If TRUE, trains on the wgpu backend. If FALSE, trains on the CPU via flex. Defaults to TRUE.

Value

A named list with two elements: embedding (numerical matrix of dimensions samples x n_dim) and model (external pointer to the trained encoder for use with rs_parametric_umap_predict).

```
rs_parametric_umap_predict
```

Predict new data using a trained parametric UMAP model

Description

[Experimental] Runs forward inference through the trained encoder network. The prediction automatically uses whichever backend (GPU or CPU) the model was trained on.

Usage

```
rs_parametric_umap_predict(model, data)
```

Arguments

model	External pointer to the trained parametric UMAP model, as returned by <code>rs_parametric_umap</code> .
data	Numerical matrix. New data of dimensions samples x features. The number of features must match the training data.

Value

Numerical matrix of dimensions samples x n_dim with the predicted embeddings.

`rs_tsne_from_knn_gpu` *tSNE implementation from a pre-computed kNN graph*

Description

[Experimental] Wraps the tSNE implementation in manifolds-rs. "fft_3k_gpu" runs the optimiser on the GPU; "bh", "bh_qd", "fft" and "fft_3k" keep it on the CPU. This version takes a pre-computed kNN graph, please see [new_nearest_neighbour\(\)](#).

Usage

```
rs_tsne_from_knn_gpu(
  embd,
  knn_data,
  n_dim,
  perplexity,
  approx_type,
  tsne_params,
  seed,
  use_high_precision,
  backend = "wgpu",
  verbose
)
```

Arguments

embd	Numerical matrix. The data to use to generate the embeddings. Should be of dimensions samples x features.
knn_data	NearestNeighbours class from R.
n_dim	Integer. Number of tSNE dimensions to return. Needs to be two, others are not supported.
perplexity	Numeric. The tSNE perplexity parameter.
approx_type	String. One of c("fft_3k_gpu", "bh", "bh_qd", "fft", "fft_3k"). Which repulsive-force approximation to use; "bh_qd" is the depth-capped quick-and-dirty Barnes-Hut.
tsne_params	Named list. List that contains all of the key parameters for the tSNE generation.

seed	Integer. Seed for reproducibility.
use_high_precision	Optional logical. Controls fp32 vs fp64. If NULL will use sensible default thresholding. Ignored for "fft_3k_gpu", which always runs in fp32.
backend	String. "wgpu" or "mlx". "mlx" runs the kNN search and the "fft_3k_gpu" optimiser on MLX (Apple Silicon, needs a build with the MLX backend, see mlx_available()); CPU optimisers stay on the CPU.
verbose	Integer. If 0L -> silent or 1L for normal verbosity; 2L for detailed verbosity.

Value

The tSNE embeddings.

rs_tsne_gpu	<i>tSNE implementation</i>
-------------	----------------------------

Description

[Experimental] Wraps the tSNE implementation in manifolds-rs. The kNN search runs on the GPU. "fft_3k_gpu" runs the optimiser on the GPU as well; "bh", "bh_qd", "fft" and "fft_3k" keep it on the CPU.

Usage

```
rs_tsne_gpu(  
  embd,  
  n_dim,  
  perplexity,  
  approx_type,  
  tsne_params,  
  seed,  
  use_high_precision,  
  backend = "wgpu",  
  verbose  
)
```

Arguments

embd	Numerical matrix. The data to use to generate the embeddings. Should be of dimensions samples x features.
n_dim	Integer. Number of tSNE dimensions to return. Needs to be two, others are not supported.
perplexity	Numeric. The tSNE perplexity parameter.
approx_type	String. One of c("fft_3k_gpu", "bh", "bh_qd", "fft", "fft_3k"). Which repulsive-force approximation to use; "bh_qd" is the depth-capped quick-and-dirty Barnes-Hut.

tsne_params	Named list. List that contains all of the key parameters for the tSNE generation.
seed	Integer. Seed for reproducibility.
use_high_precision	Optional logical. Controls fp32 vs fp64. If NULL will use sensible default thresholding. Ignored for "fft_3k_gpu", which always runs in fp32.
backend	String. "wgpu" or "mlx". "mlx" runs the kNN search and the "fft_3k_gpu" optimiser on MLX (Apple Silicon, needs a build with the MLX backend, see mlx_available()); CPU optimisers stay on the CPU.
verbose	Integer. If 0L -> silent or 1L for normal verbosity; 2L for detailed verbosity.

Value

The tSNE embeddings.

rs_umap_from_knn_gpu *UMAP implementation from a pre-computed kNN graph*

Description

[Experimental] This is the wrapper function into the Rust interface for UMAP and takes a pre-computed kNN graph, skipping the graph build. GPU acceleration applies to the optimisation.

Usage

```
rs_umap_from_knn_gpu(
  embd,
  knn_data,
  n_dim,
  min_dist,
  spread,
  k,
  umap_params,
  seed,
  use_high_precision,
  backend = "wgpu",
  verbose
)
```

Arguments

embd	Numerical matrix. The data to use to generate the embeddings. Should be of dimensions samples x features.
knn_data	NearestNeighbours class from R.
n_dim	Integer. Number of UMAP dimensions to return.
min_dist	Numeric. Minimum distance to use.

spread	Numeric. Spread parameter to use.
k	Integer. Number of nearest neighbours to consider
umap_params	Named list. List that contains all of the key parameters for the UMAP generation.
seed	Integer. Seed for reproducibility.
use_high_precision	Optional logical. Controls fp32 vs fp64 for. If NULL will use sensible default thresholding.
backend	String. "wgpu" or "mlx". "mlx" runs the kNN search and the "adam_gpu" optimiser on MLX (Apple Silicon, needs a build with the MLX backend, see mlx_available()); CPU optimisers stay on the CPU.
verbose	Integer. If 0L -> silent or 1L for normal verbosity; 2L for detailed verbosity.

Value

The UMAP embeddings.

rs_umap_gpu

UMAP implementation

Description

[Experimental] This is the wrapper function into the Rust interface for GPU-accelerated UMAP. Leverages GPU acceleration in the kNN search and also in the optimisation.

Usage

```
rs_umap_gpu(
  embd,
  n_dim,
  min_dist,
  spread,
  k,
  umap_params,
  seed,
  use_high_precision,
  backend = "wgpu",
  verbose
)
```

Arguments

embd	Numerical matrix. The data to use to generate the embeddings. Should be of dimensions samples x features.
n_dim	Integer. Number of UMAP dimensions to return.

min_dist	Numeric. Minimum distance to use.
spread	Numeric. Spread parameter to use.
k	Integer. Number of nearest neighbours to consider
umap_params	Named list. List that contains all of the key parameters for the UMAP generation.
seed	Integer. Seed for reproducibility.
use_high_precision	Optional logical. Controls fp32 vs fp64 for. If NULL will use sensible default thresholding.
backend	String. "wgpu" or "mlx". "mlx" runs the kNN search and the "adam_gpu" optimiser on MLX (Apple Silicon, needs a build with the MLX backend, see mlx_available()); CPU optimisers stay on the CPU.
verbose	Integer. If 0L -> silent or 1L for normal verbosity; 2L for detailed verbosity.

Value

The UMAP embeddings.

save_parametric_umap	<i>Save a parametric UMAP as a qs2 file</i>
----------------------	---

Description

Save a parametric UMAP as a qs2 file

Usage

```
save_parametric_umap(model, path)
```

Arguments

model	ParametricUmapModel you want to save.
path	String. The path to the file. Needs to end with ".qs" file extension.

Value

Saves the file to the provided path.

scenic_grn_sc_gpu *GPU-accelerated SCENIC GRN inference*

Description

GPU counterpart of `bixverse::scenic_grn_sc()`. Runs multi-output tree regression on the WGPU backend and returns a ScenicGrn object. Dispatches on SingleCells (disk-backed .bin counts) or MetaCells (in-memory sparse) from the bixverse package. GBM is not supported on GPU; use the CPU version for `learner_type = "grnboost2"`.

If `genes_to_take` is NULL, the CPU `bixverse::scenic_gene_filter_sc()` is used to shortlist targets (cheap min-counts / min-cells scan).

Usage

```
scenic_grn_sc_gpu(
  object,
  tf_ids,
  scenic_params = bixverse::params_scenic(),
  wave_byte_budget = 4 * 1024^3,
  genes_to_take = NULL,
  cells_to_take = NULL,
  streaming = NULL,
  random_seed = 42L,
  .verbose = TRUE
)
```

Arguments

<code>object</code>	SingleCells or MetaCells class from bixverse.
<code>tf_ids</code>	Character vector. Gene identifiers of the transcription factors to use as predictors.
<code>scenic_params</code>	Named list. See <code>bixverse::params_scenic()</code> .
<code>wave_byte_budget</code>	Numeric. VRAM ceiling for per-wave histogram and cumulative tensors (bytes). Default 4 GiB. Shrink on tight VRAM budgets, raise on 16 GB+ adapters to let the scheduler pick a wider wave.
<code>genes_to_take</code>	Optional character vector. Target genes to include. If NULL, the CPU gene filter runs first.
<code>cells_to_take</code>	Optional character vector. Cell names to include. If NULL, all filtered cells are used.
<code>streaming</code>	Optional boolean. Only used on SingleCells. If TRUE, the streaming GPU driver is used (bounded host memory). If NULL, is auto-picked from cell count via bixverse's internal <code>auto_streaming</code> . Ignored for MetaCells.
<code>random_seed</code>	Integer. For reproducibility.
<code>.verbose</code>	Boolean or integer. Controls verbosity. FALSE -> quiet, TRUE or 1L -> normal, 2L -> detailed.

Value

A ScenicGrn object with the gene x TF importance matrix.

References

Aibar et al., Nat Methods, 2017.

scrublet_gpu_sc

Doublet detection with Scrublet on the GPU

Description

GPU counterpart of `bixverse::scrublet_sc()`. Three stages run on the WGPU backend: the randomised sparse SVD of the observed cells, the projection of the simulated doublets into that PC space, and the kNN over the combined embedding. HVG selection, doublet simulation, the kNN classifier and the Otsu threshold stay on the CPU, so the speedup tracks how much of the run the SVD and the kNN own. That share grows with cell count: the combined embedding is $(1 + \text{sim_doublet_ratio}) * \text{n_cells}$ rows tall and an exhaustive kNN over it is quadratic.

Usage

```
scrublet_gpu_sc(
  object,
  scrublet_params = params_scrublet_gpu(),
  seed = 42L,
  streaming = NULL,
  cells_to_use = NULL,
  group_by = NULL,
  return_combined_pca = FALSE,
  return_pairs = FALSE,
  .verbose = TRUE
)
```

Arguments

object	SingleCells class from bixverse.
scrublet_params	List. Output of <code>params_scrublet_gpu()</code> .
seed	Integer. Random seed.
streaming	Optional boolean. Shall the counts be streamed during HVG selection. If NULL, resolved from the cell count.
cells_to_use	Optional character vector. Names of the cells to run on. The returned object covers exactly these cells.
group_by	Optional string. Column in the obs table to run the method per level of, typically a sample identifier.

return_combined_pca	Boolean. Shall the combined PCA of observed cells and simulated doublets be returned.
return_pairs	Boolean. Shall the parent indices of the simulated doublets be returned.
.verbose	Boolean or integer. Controls verbosity and returns run times. FALSE -> quiet, TRUE or 1L -> normal verbosity, 2L -> detailed verbosity.

Details

Scores do not match the CPU bit for bit. The SVD is randomised on both sides but draws a different sketch, and the GPU indices break neighbour ties differently. Expect a correlation around 0.99 rather than equality, and a handful of borderline calls to flip because Otsu's threshold is a step function of the histogram bins.

Value

A ScrubletRes S3 object, identical in shape to the CPU one, with the following items:

- predicted_doublets - Boolean vector indicating which observed cells were predicted as doublets (TRUE = doublet, FALSE = singlet).
- doublet_scores_obs - Numerical vector with the likelihood of being a doublet for the observed cells.
- doublet_scores_sim - Numerical vector with the likelihood of being a doublet for the simulated cells.
- doublet_errors_obs - Numerical vector with the standard errors of the scores for the observed cells.
- z_scores - Z-scores for the observed cells. Represents: $\text{score} - \text{threshold} / \text{error}$.
- threshold - Used threshold.
- detected_doublet_rate - Fraction of cells that are called as doublet.
- detectable_doublet_fraction - Fraction of simulated doublets with scores above the threshold.
- overall_doublet_rate - Estimated overall doublet rate.
- pca - Optional PCA embeddings across the original cells and simulated doublets.
- pair_1 - Optional index of the parent cell 1 of the simulated doublets.
- pair_2 - Optional index of the parent cell 2 of the simulated doublets.

The 0-indexed cell indices are attached as the cell_indices attribute. Grouped runs additionally carry grouped and group_by_col attributes and a cell_groups element.

References

Wolock, et al., Cell Syst, 2020

stabilised_nmf_gpu_sc *Run stabilised (multi-run) NMF on the GPU over single cell or meta cell data*

Description

GPU counterpart of `bixverse::stabilised_nmf_sc()`. Runs `n_runs` HALS NMF with random initialisations seeded by `seed + i`. The `nmf_init` field in `nmf_hals_params` is ignored; random init is always used.

The counts upload once and serve every restart, but the restarts themselves run one after the other on the single device, where the CPU version spreads them across cores. On a small matrix the CPU can still win.

Usage

```
stabilised_nmf_gpu_sc(
  object,
  k,
  cell_ids = NULL,
  gene_ids = NULL,
  preprocessing = "none",
  use_second_layer = TRUE,
  nmf_hals_params = bixverse::params_nmf_hals(),
  n_runs = 30L,
  seed = 42L,
  backend = c("wgpu", "mlx"),
  .verbose = TRUE
)
```

Arguments

<code>object</code>	SingleCells or MetaCells class from <code>bixverse</code> .
<code>k</code>	Integer. Number of latent factors per run. At most 128, see <code>NMF_GPU_MAX_RANK</code> .
<code>cell_ids</code>	Optional character. Cell ids (or meta cell ids) to restrict the NMF to. If NULL, uses <code>bixverse::get_cells_to_keep()</code> for SingleCells and all meta cells for MetaCells.
<code>gene_ids</code>	Optional character. Gene ids to restrict the NMF to. If NULL, uses <code>bixverse::get_hvg()</code> on the object.
<code>preprocessing</code>	String. One of <code>c("none", "sd", "sqrt_sd")</code> .
<code>use_second_layer</code>	Boolean. If TRUE, runs NMF on the normalised counts (recommended); if FALSE, on the raw counts.
<code>nmf_hals_params</code>	List, see <code>bixverse::params_nmf_hals()</code> .
<code>n_runs</code>	Integer. Number of random restarts.

seed	Integer. Random seed for initialisation.
backend	String. One of c("wgpu", "mlx"). "mlx" runs the HALS solves on Apple Silicon via MLX and needs a build with the MLX backend, see mlx_available() .
.verbose	Boolean or integer. Controls verbosity. FALSE -> quiet, TRUE or 1L -> normal verbosity, 2L -> detailed verbosity.

Value

A StabilisedNmfResult object, the same class `bixverse::stabilised_nmf_sc()` returns.

tsne_gpu	<i>Rust-based t-SNE (GPU)</i>
----------	-------------------------------

Description

Performs t-SNE dimensionality reduction on the input data. The kNN search runs on the GPU. With the default "fft_3k_gpu" the optimiser runs on the GPU as well, so the whole embedding stays on the device; the other approximations optimise on the CPU.

Usage

```
tsne_gpu(  
  data,  
  knn = NULL,  
  n_dim = 2L,  
  perplexity = 20,  
  approx_type = c("fft_3k_gpu", "bh", "bh_qd", "fft", "fft_3k"),  
  knn_method = c("ivf", "exhaustive", "nn descent"),  
  nn_params = params_nn_gpu(),  
  tsne_params = params_tsne_gpu(),  
  seed = 42L,  
  use_high_precision = NULL,  
  backend = c("wgpu", "mlx"),  
  .verbose = TRUE  
)
```

Arguments

data	Numerical matrix or data frame. The data to embed of shape samples x features. Will be coerced to a matrix.
knn	Optional NearestNeighbours class. If provided, t-SNE will skip the k-nearest neighbour graph generation and use this one. Defaults to NULL. See manifoldsR::new_nearest_neighbo for details.
n_dim	Integer. Number of dimensions in the embedding space. Currently only 2L is supported. Defaults to 2L.

perplexity	Numeric. Perplexity parameter, related to the number of nearest neighbours used in manifold learning. Typical values are between 5 and 50. Defaults to 20.0.
approx_type	Character. Approximation method for computing repulsive forces. One of "fft_3k_gpu" (three-kernel FFT interpolation on the GPU, the default), "bh" for Barnes-Hut, "bh_qd" for the quick-and-dirty Barnes-Hut of qdtsne (tree depth capped at max_depth in params_tsne_gpu() , faster and coarser), "fft" for FFT-accelerated interpolation or "fft_3k" for its three-kernel variant (one forward and three inverse FFTs per epoch instead of four each). The last four run on the CPU.
knn_method	Character. GPU-accelerated (approximate) nearest neighbour method to use. One of "ndescent", "exhaustive", or "ivf".
nn_params	Named list. Nearest neighbour search parameters, see params_nn_gpu() . tSNE uses higher k usually, hence, "ivf" is the default here.
tsne_params	Named list. t-SNE (GPU) algorithm parameters, see params_tsne_gpu() .
seed	Integer. Random seed for reproducibility. Defaults to 42L.
use_high_precision	Optional boolean. Gives fine-grained control over fp32 vs fp64 usage. The GPU kNN calculations will be forced into fp32. Ignored with a warning for "fft_3k_gpu", which always runs in fp32.
backend	String. One of c("wgpu", "mlx"). "mlx" runs the kNN search and the "fft_3k_gpu" optimiser on MLX (Apple Silicon, needs a build with the MLX backend, see mlx_available()). CPU optimisers stay on the CPU.
.verbose	Logical. Controls verbosity. Defaults to TRUE.

Details

The number of neighbours is derived from perplexity on the Rust side following the usual $3 * \text{perplexity}$ convention.

Value

A numerical matrix with dimensions samples x n_dim containing the t-SNE embedding.

tsne_gpu_sc

Run t-SNE on a SingleCells object (GPU)

Description

GPU-accelerated counterpart to [bixverse::tsne_sc\(\)](#). Runs [tsne_gpu\(\)](#) on an embedding pulled from the object. The kNN runs on the GPU; with the default "fft_3k_gpu" the optimiser does too. t-SNE derives the number of neighbours from perplexity on the Rust side (the usual $3 * \text{perplexity}$ convention). To avoid a silent mismatch with the cached kNN, use_knn defaults to FALSE: every call generates a fresh GPU kNN sized to the requested perplexity. Handy for sweeping perplexities since the kNN is cheap on GPU.

Usage

```
tsne_gpu_sc(
  object,
  use_knn = FALSE,
  embd_to_use = "pca",
  slot_name = "tsne",
  no_embd_to_use = NULL,
  modality = c("rna", "adt", "wnn"),
  n_dim = 2L,
  perplexity = 20,
  approx_type = c("fft_3k_gpu", "bh", "bh_qd", "fft", "fft_3k"),
  knn_method = c("ivf", "exhaustive", "nn descent"),
  nn_params = params_nn_gpu(),
  tsne_params = params_tsne_gpu(),
  seed = 42L,
  use_high_precision = NULL,
  backend = c("wgpu", "mlx"),
  .verbose = TRUE
)
```

Arguments

<code>object</code>	SingleCells (or SingleCellsMultiModal) class.
<code>use_knn</code>	Boolean. Defaults to FALSE. Set to TRUE to reuse the cached kNN; only sensible when the stored k is at least 3 * perplexity.
<code>embd_to_use</code>	String. The embedding to use for t-SNE. Must be available in the object for the chosen modality.
<code>slot_name</code>	String. The name of this embedding within the object. Defaults to "tsne".
<code>no_embd_to_use</code>	Optional integer. Number of embedding dimensions to use. If NULL, all will be used.
<code>modality</code>	String. On which modality to run t-SNE. One of c("rna", "adt", "wnn"). The two latter options are only available on SingleCellsMultiModal.
<code>n_dim</code>	Integer. Number of t-SNE dimensions. Currently only 2L is supported. Defaults to 2L.
<code>perplexity</code>	Numeric. Perplexity parameter. Typical values between 5 and 50. Defaults to 20.0.
<code>approx_type</code>	String. Approximation method. One of "fft_3k_gpu" (three-kernel FFT on the GPU, the default), "bh" (Barnes-Hut), "bh_qd" (quick-and-dirty Barnes-Hut, depth-capped tree), "fft" or "fft_3k" (CPU FFT interpolation, four or three kernels). See tsne_gpu() .
<code>knn_method</code>	String. GPU (approximate) nearest neighbour method. One of c("ivf", "exhaustive", "nn descent"). Default is "ivf" here, as it deals usually better with the high k in tSNE.
<code>nn_params</code>	Named list. GPU kNN parameters, see params_nn_gpu() .
<code>tsne_params</code>	Named list. t-SNE (GPU) parameters, see params_tsne_gpu() .

seed	Integer. For reproducibility.
use_high_precision	Optional boolean. Fine-grained fp32 vs fp64 control. GPU kNN is always fp32. Ignored for "fft_3k_gpu".
backend	String. One of c("wgpu", "mlx"). "mlx" moves the kNN search and the "fft_3k_gpu" optimiser to MLX, see tsne_gpu() .
.verbose	Boolean or integer. Controls verbosity.

Value

The object with a "tsne" embedding added. If the requested embedding is missing, returns the object unchanged with a warning.

See Also

[tsne_gpu\(\)](#), [bixverse::tsne_sc\(\)](#), [umap_gpu_sc\(\)](#)

umap_gpu

Rust-based UMAP (GPU)

Description

Performs UMAP dimensionality reduction on the input data. This function provides a user-friendly interface with input validation before calling the Rust implementation. Leverages GPU-accelerated kNN searches and in the default setting also uses a GPU-accelerated Adam optimiser for the embedding.

Usage

```
umap_gpu(
  data,
  knn = NULL,
  n_dim = 2L,
  k = 15L,
  min_dist = 0.5,
  spread = 1,
  knn_method = c("nn descent", "exhaustive", "ivf"),
  nn_params = params_nn_gpu(),
  umap_params = params_umap_gpu(),
  seed = 42L,
  use_high_precision = NULL,
  backend = c("wgpu", "mlx"),
  .verbose = TRUE
)
```

Arguments

data	Numerical matrix or data frame. The data to embed of shape samples x features. Will be coerced to a matrix.
knn	Optional NearestNeighbours class. If provided, UMAP will skip the k-nearest neighbour graph generation and use this one. Defaults to NULL. See manifoldsR::new_nearest_neighbours for details.
n_dim	Integer. Number of dimensions in the embedding space. Defaults to 2L.
k	Integer. Number of nearest neighbours to consider for manifold approximation. Larger values result in more global structure being preserved. Defaults to 15L.
min_dist	Numeric. Minimum distance between points in the embedding. Controls how tightly points are packed. Smaller values result in more clustered embeddings. Must be >= 0. Defaults to 0.5. If you use SGD, consider reducing this!
spread	Numeric. Effective scale of embedded points. Determines the scale at which embedded points will be spread out. Defaults to 1.0.
knn_method	Character. (Approximate) Nearest neighbour method to use. One of "exhaustive", "ivf" or "nn descent". These are GPU-accelerated methods.
nn_params	Named list. Nearest neighbour search parameters, see params_nn_gpu() .
umap_params	Named list. UMAP (GPU) algorithm parameters, see params_umap_gpu() .
seed	Integer. Random seed for reproducibility. Defaults to 42L.
use_high_precision	Optional boolean. Gives fine-grained control over fp32 vs fp64 usage. The GPU calculations will be forced into fp32.
backend	String. One of c("wgpu", "mlx"). "mlx" runs the kNN search and the "adam_gpu" optimiser on MLX (Apple Silicon, needs a build with the MLX backend, see mlx_available()). CPU optimisers stay on the CPU.
.verbose	Logical. Controls verbosity. Defaults to TRUE.

Value

A numerical matrix with dimensions samples x n_dim containing the UMAP embedding.

umap_gpu_sc	<i>Run UMAP on a SingleCells object (GPU)</i>
-------------	---

Description

GPU-accelerated counterpart to [bixverse::umap_sc\(\)](#). Pulls an embedding (defaulting to PCA) off the object, runs [umap_gpu\(\)](#) on it (GPU kNN plus GPU Adam optimiser by default), and writes the resulting embedding back into `sc_cache$other_embeddings[[slot_name]]`.

When `use_knn = TRUE` (the default), the kNN graph already stored on the object is reused via [bixverse::sc_knn_to_nearest_neighbours\(\)](#). Otherwise a fresh GPU kNN is built from the chosen embedding.

Usage

```
umap_gpu_sc(
  object,
  use_knn = TRUE,
  embd_to_use = "pca",
  slot_name = "umap",
  no_embd_to_use = NULL,
  modality = c("rna", "adt", "wnn"),
  n_dim = 2L,
  k = 15L,
  min_dist = 0.5,
  spread = 1,
  knn_method = c("nn descent", "exhaustive", "ivf"),
  nn_params = params_nn_gpu(),
  umap_params = params_umap_gpu(),
  seed = 42L,
  use_high_precision = NULL,
  backend = c("wgpu", "mlx"),
  .verbose = TRUE
)
```

Arguments

<code>object</code>	SingleCells (or SingleCellsMultiModal) class.
<code>use_knn</code>	Boolean. Use the kNN graph found in the object. Defaults to TRUE. Only reused if the modality lines up; otherwise a fresh GPU kNN is generated.
<code>embd_to_use</code>	String. The embedding to use for UMAP. Must be available in the object for the chosen modality.
<code>slot_name</code>	String. The name of this embedding within the object. Defaults to "umap".
<code>no_embd_to_use</code>	Optional integer. Number of embedding dimensions to use. If NULL, all will be used.
<code>modality</code>	String. On which modality to run UMAP. One of <code>c("rna", "adt", "wnn")</code> . The two latter options are only available on <code>SingleCellsMultiModal</code> .
<code>n_dim</code>	Integer. Number of UMAP dimensions. Defaults to 2L.
<code>k</code>	Integer. Number of nearest neighbours. Defaults to 15L.
<code>min_dist</code>	Numeric. Minimum distance between embedded points. Defaults to 0.5.
<code>spread</code>	Numeric. Effective scale of embedded points. Defaults to 1.0.
<code>knn_method</code>	String. GPU (approximate) nearest neighbour method. One of <code>c("nn descent", "exhaustive", "ivf")</code> .
<code>nn_params</code>	Named list. GPU kNN parameters, see params_nn_gpu() .
<code>umap_params</code>	Named list. UMAP (GPU) parameters, see params_umap_gpu() .
<code>seed</code>	Integer. For reproducibility.
<code>use_high_precision</code>	Optional boolean. Fine-grained fp32 vs fp64 control for the optimiser. GPU kNN is always fp32.

backend	String. One of <code>c("wgpu", "mlx")</code> . "mlx" runs the kNN search and the "adam_gpu" optimiser on MLX, see umap_gpu() .
.verbose	Boolean or integer. Controls verbosity.

Value

The object with a "umap" embedding added. If the requested embedding is missing, returns the object unchanged with a warning.

See Also

[umap_gpu\(\)](#), [bixverse::umap_sc\(\)](#), [tsne_gpu_sc\(\)](#)

Index

bbknn_gpu_sc, [3](#)
bixverse::bbknn_sc(), [3](#)
bixverse::bonsai_sc(), [4, 5](#)
bixverse::consensus_nmf_sc(), [6, 7](#)
bixverse::fast_cluster_sc(), [8](#)
bixverse::fast_mnn_sc(), [9, 10](#)
bixverse::find_clusters_sc(), [10](#)
bixverse::find_hvg_batch_aware_sc(),
[10](#)
bixverse::generate_seacells_sc(), [15](#)
bixverse::get_cells_to_keep(), [7, 25, 26,](#)
[63](#)
bixverse::get_hvg(), [6, 7, 25, 26, 63](#)
bixverse::install_agent_skill(), [18](#)
bixverse::leiden_clustering(), [20](#)
bixverse::MetaCells(), [16](#)
bixverse::nebula_sc(), [22](#)
bixverse::nmf_k_sweep_sc(), [25](#)
bixverse::nmf_sc(), [24, 25](#)
bixverse::params_hvg_defaults(), [43](#)
bixverse::params_knn_defaults(), [16, 29,](#)
[44](#)
bixverse::params_knn_graph_defaults(),
[12, 38](#)
bixverse::params_nebula(), [30](#)
bixverse::params_nmf_consensus(), [7, 26](#)
bixverse::params_nmf_hals(), [7, 25, 26,](#)
[63](#)
bixverse::params_norm_doublets_defaults(),
[43](#)
bixverse::params_sc_bbknn(), [35](#)
bixverse::params_sc_bonsai(), [5](#)
bixverse::params_sc_fast_cluster(), [36](#)
bixverse::params_sc_fastmnn(), [39](#)
bixverse::params_sc_pca(), [6](#)
bixverse::params_sc_seacells(), [16](#)
bixverse::params_scenic(), [60](#)
bixverse::params_scrublet(), [43](#)
bixverse::sc_knn_to_nearest_neighbours(),
[68](#)
bixverse::scenic_gene_filter_sc(), [60](#)
bixverse::scenic_grn_sc(), [60](#)
bixverse::scrublet_sc(), [61](#)
bixverse::stabilised_nmf_sc(), [63, 64](#)
bixverse::tsne_sc(), [65, 67](#)
bixverse::umap_sc(), [68, 70](#)
bonsai_gpu_sc, [4](#)
calculate_pca_gpu_sc, [5](#)
consensus_nmf_gpu_sc, [6](#)
consensus_nmf_gpu_sc(), [24, 25](#)
fast_cluster_gpu_sc, [8](#)
fast_mnn_gpu_sc, [9](#)
find_clusters_gpu_sc, [10](#)
find_neighbours_gpu_sc, [11](#)
find_neighbours_sc(), [11](#)
generate_gpu_knn_sc, [13](#)
generate_gpu_knn_sc(), [11](#)
generate_knn_graph_gpu, [14](#)
generate_knn_sc(), [13](#)
generate_seacells_gpu_sc, [15](#)
gpu_available, [17](#)
harmony_v2_gpu_sc, [17](#)
install_agent_skill_gpu, [18](#)
k_means_cluster_gpu, [19](#)
leiden_clustering_gpu, [20](#)
leiden_clustering_gpu(), [10, 11](#)
load_parametric_umap, [21](#)
manifoldsR::new_nearest_neighbour(),
[64, 68](#)
mlx_available, [22](#)
mlx_available(), [6, 7, 9, 12, 14–16, 18, 25,](#)
[26, 48–50, 52, 56–59, 64, 65, 68](#)

- nebula_gpu_sc, [22](#)
- new_nearest_neighbour(), [55](#)
- NMF_GPU_MAX_RANK, [7](#), [25](#), [26](#), [63](#)
- nmf_gpu_sc, [24](#)
- nmf_k_sweep_gpu_sc, [25](#)
- nmf_k_sweep_gpu_sc(), [6](#), [24](#)
- parametric_umap, [27](#)
- params_kmeans_gpu, [28](#)
- params_kmeans_gpu(), [20](#)
- params_knn_gpu_defaults, [29](#)
- params_knn_gpu_defaults(), [35](#), [36](#), [39](#), [40](#), [44](#)
- params_nebula_gpu, [30](#)
- params_nebula_gpu(), [22](#), [23](#)
- params_nn(), [27](#)
- params_nn_gpu, [32](#)
- params_nn_gpu(), [12](#), [13](#), [15](#), [50](#), [65](#), [66](#), [68](#), [69](#)
- params_parametric_umap, [34](#)
- params_parametric_umap(), [27](#)
- params_sc_bbknn_gpu, [35](#)
- params_sc_bbknn_gpu(), [4](#)
- params_sc_fast_cluster_gpu, [36](#)
- params_sc_fast_cluster_gpu(), [8](#)
- params_sc_fastmnn_gpu, [39](#)
- params_sc_fastmnn_gpu(), [10](#), [49](#)
- params_sc_harmony_v2_gpu, [40](#)
- params_sc_harmony_v2_gpu(), [18](#)
- params_sc_pca(), [5](#)
- params_scrublet_gpu, [43](#)
- params_scrublet_gpu(), [61](#)
- params_tsne_gpu, [44](#)
- params_tsne_gpu(), [65](#), [66](#)
- params_umap_gpu, [46](#)
- params_umap_gpu(), [68](#), [69](#)
- predict.ParametricUmapModel, [47](#)
- rs_cor_gpu, [48](#)
- rs_cov_gpu, [49](#)
- rs_fast_mnn_gpu, [49](#)
- rs_gpu_available, [50](#)
- rs_gpu_knn, [50](#)
- rs_gpu_plane_ops, [51](#)
- rs_harmony_v2_gpu, [51](#)
- rs_kmeans_gpu, [52](#)
- rs_leiden_clustering_gpu, [53](#)
- rs_parametric_umap, [53](#)
- rs_parametric_umap_predict, [54](#)
- rs_tsne_from_knn_gpu, [55](#)
- rs_tsne_gpu, [56](#)
- rs_umap_from_knn_gpu, [57](#)
- rs_umap_gpu, [58](#)
- save_parametric_umap, [59](#)
- scenic_grn_sc_gpu, [60](#)
- scrublet_gpu_sc, [61](#)
- stabilised_nmf_gpu_sc, [63](#)
- stabilised_nmf_gpu_sc(), [6](#)
- tsne_gpu, [64](#)
- tsne_gpu(), [65–67](#)
- tsne_gpu_sc, [65](#)
- tsne_gpu_sc(), [70](#)
- umap_gpu, [67](#)
- umap_gpu(), [68](#), [70](#)
- umap_gpu_sc, [68](#)
- umap_gpu_sc(), [67](#)