Package: SigBridgeR (via r-universe)

November 28, 2026

Title Integrative Toolkit for Linking Phenotypes to Cell Subpopulations via scRNA-seq and Bulk Data

Version 4.0.0

Description Identifies phenotype-associated cell subpopulations by integrating phenotypic data (e.g., survival, drug sensitivity), bulk gene expression, and single-cell RNA-seq (scRNA-seq) profiles. It employs multiple algorithms to reliably connect cellular features with clinical or functional phenotypes, offering a unified pipeline for multi-modal analysis that uncovers biologically and clinically relevant cell states in heterogeneous datasets.

License GPL (>= 3)

Encoding UTF-8

URL https://wanglabcsu.github.io/sigbridger

BugReports https://github.com/WangLabCSU/SigBridgeR/issues

Roxygen list(markdown = TRUE)

Imports beachmat, chk, cli (>= 3.0.0), data.table (>= 1.9.8), ggplot2, lifecycle, Matrix, purrr, Rcpp, reticulate, rlang (>= 1.0.0), S7 (>= 1.0.0), Seurat (>= 5.0.0), SeuratObject (>= 5.0.0), SigBridgeRUtils (>= 0.2.5)

Depends R (>= 4.2.0)

Suggests anndata, anndataR, AUCell, bench, callr, carrier, celldex, cheapr, codetools, covr, curl, DEGAS (>= 1.0.1), DEGASv2, DESeq2, doclisting, dplyr, edgeR, fgsea, furrr, future, future.mirai, GenomicRanges, ggforce, ggupset, ggVennDiagram, glue, GSVA, here, IDConverter, jsonlite, KernSmooth, knitr, lintr, LPSGL, matrixStats, matrixTests, methods, microbenchmark, mLLMCelltype, org.Hs.eg.db, pak, parallel, patchwork, PhenoMapR, PIPET, preprocessCore, qs2, RColorBrewer, rmarkdown, rSIDISH (>= 0.0.2), rstudioapi, rTiRank, rtracklayer, scAB (>= 1.0.4), scCustomize, SCIPAC, Scissor (>= 2.0.1), scPAS (>= 0.2.1), ScPP, SingleCellExperiment, SingleR,

sparseMatrixStats, spelling, testthat (>= 3.0.0), tibble, tictoc, tidydr, tidyr, UCell, WGCNA, withr, zeallot

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first watcher, parallel*

LinkingTo Rcpp, RcppArmadillo (>= 0.12.4.0.0), assorthead, beachmat

SystemRequirements C++17, zlib (>= 1.2.4)

Config/roxygen2/version 8.1.1

Language en-US

Copyright Duanchen Sun (for the Scissor code included in R/vendor-Scissor.R), Travis Johnson and Zhi Huang (for the DEGAS code included in R/vendor-DEGAS.R), Qinran Zhang (for the scAB code included in R/vendor-scAB.R), Hongmei Zhang (for the LP_SGL code included in R/vendor-LP_SGL.R), Dailin Gan and Jun Li (for the SCIPAC code included in R/vendor-SCIPAC.R), Xinjia Ruan (for the PIPET code included in R/vendor-PIPET.R), Aimin Xie (for the scPAS code included in R/vendor-scPAS.R), Xiaosheng Wang (for the scPP code included in R/vendor-scPP.R), Yasmin Jolasun and Jun Ding (for the SIDISH code included in R/vendor-SIDISH.R), Yuxiang Lin (for the TiRank code included in R/vendor-TiRank.R)

Config/pak/sysreqs
libglpk-dev make libicu-dev libpng-dev libxml2-dev libssl-dev python3 zlib1g-dev

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

Date/Publication 2026-11-27 23:52:22 UTC

RemoteUrl https://github.com/WangLabCSU/SigBridgeR

RemoteRef HEAD

RemoteSha 7015f9ca6f2538d273eae2e35da53871db11998f

Contents

AddMetaFeature 4
AddMisc 5
aggregate-dups 6
AnnotationMethod 8
autoplot-SigBridgeR 9
BenchAcc 10
BenchAccConfig 12
BenchConfig 13
BenchGeneScore 15
BenchGeneScoreConfig 17
BenchGeneScoreConfigSet 19

BenchGSEA . . . 20
BenchGSEAConfig . . . 22
BenchGSEAConfigSet . . . 23
BenchMemory . . . 24
BenchMemoryConfig . . . 26
BenchMemoryConfigSet . . . 27
CacheSetHere . . . 29
CacheSysCall . . . 31
CellTypistAnnotate . . . 33
CheckCache . . . 36
CheckNA . . . 37
ChooseCache . . . 39
ChooseNormalization . . . 40
CountsToTPM . . . 43
DoDEGAS . . . 44
DoDEGASv2 . . . 47
DoLP_SGL . . . 50
DoPhenoMapR . . . 53
DoPIPET . . . 56
DoscAB . . . 58
DoSCIPAC . . . 61
DoScissor . . . 64
DoscPAS . . . 67
DoscPP . . . 69
DoSIDISH . . . 72
DoTiRank . . . 74
DownloadCellTypistModels . . . 77
DrawBulkPCA . . . 78
FindRobustElbow . . . 80
FPKMToTPM . . . 81
getFuncOption . . . 82
GetGeneLength . . . 84
InterceptStrategy . . . 85
IsCountsMatrix . . . 86
IsSkewedDynamic . . . 87
list_to_chr . . . 90
list_to_chr_formated . . . 91
ListCellTypistModels . . . 92
ListPyEnv . . . 93
LoadCache . . . 94
LoadRefData . . . 95
MergeResult . . . 96
Method . . . 99
MethodParamSet . . . 100
mLLMCelltypeAnnotate . . . 101
palette_SigBridgeR . . . 105
Pattern2Colname . . . 105
PhenoMap . . . 106

PhenoPreProcess . . . 107
plus-methods . . . 110
Property-visitor . . . 111
QCFilter . . . 111
QCPatternDetect . . . 112
Register . . . 113
RegisterAnnoMethod . . . 114
RegisterScreenMethod . . . 115
RegisterSeuratMethod . . . 117
SCAnnotate . . . 119
SCAnnotateStrategy . . . 120
SCPreProcess . . . 121
SCPreProcessStrategy . . . 124
Screen . . . 125
ScreenFractionPlot . . . 132
ScreenMethod . . . 134
ScreenMethodCache . . . 135
ScreenMethodConfig . . . 137
ScreenMethodResult . . . 138
ScreenStrategy . . . 139
ScreenUpset . . . 140
setFuncOption . . . 142
SetThreads . . . 143
SetupPyEnv . . . 145
SigBridgeR-properties . . . 146
SigBridgeR-S3 . . . 148
SigBridgeRBase . . . 150
SingleRAnnotate . . . 151
SymbolConvert . . . 153
TemplateScreenFunc . . . 154
ValidateScreenFunc . . . 155
WeightedVote . . . 156
WriteCache . . . 157
WriteCacheMeta . . . 159

Index **161**

`AddMetaFeature` *Add Gene-Level Metadata to Seurat Object (Vectorized, ...-based)*

Description

Add multiple feature-level metadata (vectors or 2D tables) to a Seurat object. Handles vectors (length-checked), matrices, data.frames, data.tables, etc. Columns/variables with duplicated names are suffixed (e.g., "type_1"). Gene alignment is auto-detected: rows or cols must match nrow(seurat_obj).

Usage

```
AddMetaFeature(seurat_obj, ..., assay = "RNA")
```

Arguments

seurat_obj	A Seurat object.
...	One or more metadata inputs: • Named/unnamed vectors (length = ngenes) • 2D objects (matrix, data.frame, etc.) where one dimension (rows or cols) has size = ngenes.
assay	Assay name (default: "RNA", fallback to first).

Value

Modified seurat_obj (invisibly).

AddMisc *Safely Add Miscellaneous Data to Seurat Object*

Description

Adds arbitrary data to the @misc slot of a Seurat object with automatic key conflict resolution. If the key already exists, automatically appends a numeric suffix to ensure unique key naming (e.g., "mykey_1", "mykey_2").

Usage

```
AddMisc(seurat_obj, ..., cover = TRUE)
```

Arguments

seurat_obj	A Seurat object to modify
...	key-value pairs to add to the @misc slot.
cover	Logical indicating whether to overwrite existing data. If (default TRUE).

Value

The modified Seurat object with added @misc data. The original object structure is preserved with no other modifications.

Key Generation Rules

1. If key doesn't exist: uses as-is
2. If key exists: appends the next available number (e.g., "key_1", "key_2")
3. If numbered keys exist (e.g., "key_2"): increments the highest number

Examples

```
## Not run:
# Basic usage
seurat_obj <- AddMisc(seurat_obj, "QC_stats" = qc_df)

# Auto-incrementing example
seurat_obj <- AddMisc(seurat_obj, markers = markers1)
seurat_obj <- AddMisc(seurat_obj, markers = markers2, cover=FALSE)
# Stores as "markers" and "markers_1"


## End(Not run)
```

---

aggregate-dups *Aggregate Rows or Columns with Duplicate Names*

Description

These functions collapse duplicated row names (e.g., gene symbols) or column names (e.g., sample IDs) in matrix-like objects by aggregating values using configurable methods. They support:

- **Rows**: `AggregateDupRows`: merges rows sharing the same row name.
- **Columns**: `AggregateDupCols`: merges columns sharing the same column name.
- **Both**: `AggregateDups`: convenience wrapper applying row-then-column aggregation. Designed for expression matrices, count tables, or any numeric data where feature/sample duplication occurs. Handles `matrix`, `data.frame`, and S4 `Matrix` classes (e.g. `dgCMatrix`) robustly.

Convenience wrapper that first aggregates duplicated rows, then duplicated columns. Useful for cleaning matrices where both feature and sample duplication may occur.

Usage

```
AggregateDupRows(
  x,
  method = c("max", "sum", "mean", "median", "first"),
  verbose = TRUE,
  ...
)

AggregateDupCols(
  x,
  method = c("max", "sum", "mean", "median", "first"),
  verbose = TRUE,
  ...
)
```

```
AggregateDups(
  x,
  method = c("max", "sum", "mean", "median", "first"),
  row_method = method,
  col_method = method,
  verbose = TRUE,
  ...
)
```

Arguments

x	A numeric matrix-like object (see Details).
method	Character scalar. Aggregation method (see Methods below).
verbose	Whether to print messages
...	No usage
row_method	Aggregation method for rows. Defaults to method.
col_method	Aggregation method for columns. Defaults to method.

Value

An aggregated object of the same effective type as x, with unique row/column names.

Methods

Supported methods (applied column-wise for rows, row-wise for columns):

- "max": Maximum value per group (default).
- "sum": Sum of values per group.
- "mean": Arithmetic mean (uses na.rm = TRUE).
- "median": Median value.
- "first": First occurrence in original order.

Input Types and Return Types

Input class	Output class (unless noted)
matrix	matrix
data.frame	data.frame
S4 Matrix	matrix (dense) - S4 attributes dropped for generality

Row/column order in output follows *first occurrence* of each unique name in rownames(x) / colnames(x).

Examples

```
# Full deduplication in one step
mat <- matrix(1:16, nrow = 4,
              dimnames = list(c("TP53", "TP53", "BRCA1", "ACTB"),
                              c("S1", "S1", "S2", "S3")))
mat

AggregateDups(mat, method = "sum")
```

`AnnotationMethod` *AnnotationMethod: Annotation Method Class for Cell Type Identification*

Description

`AnnotationMethod` is the S7 class for cell type annotation methods. It extends Method with Python environment support, enabling annotation methods that depend on Python-based tools (e.g., SingleR, scArches).

Usage

```
AnnotationMethod(
  method_name = character(),
  executor = function() {
     NULL
 },
  pkg_name = character(),
  method_version = NULL,
  py_env = NULL,
  sigbridger_version = get_pkg_version()
)
```

Arguments

`method_name`	`character`. A string identifying the method (e.g., `"Scissor"`, `"scPAS"`).
`executor`	`function`. The function that performs the actual computation. This is the core workhorse of each method.
`pkg_name`	The name of the R (Python) package when using this annotation method. When `method_version` is `NULL`, the method version is resolved from the R package version according to `pkg_name`.
`method_version`	`character`. A string indicating the version of the underlying software package.
`py_env`	`character` or `NULL`. A conda environment name or Python path for Python-based annotation tools. When `NULL` (default), the method version is resolved from an R package.
`sigbridger_version`	Package Version of SigBridgeR

Details

This class inherits from Method and adds a py_env property for specifying a conda environment or Python path. When py_env is NULL, the method version is resolved from an R package via r_pkg_version(). When a Python environment is specified, the version is resolved via py_pkg_version().

Value

An AnnotationMethod S7 object.

See Also

Other S7-Classes: BenchAccConfig(), BenchConfig(), BenchGSEAConfig(), BenchGeneScoreConfig(), BenchMemoryConfig(), ScreenMethodCache(), ScreenMethodConfig(), ScreenMethodResult()

---

autoplot-SigBridgeR *autoplot method for Seurat UMAP*

Description

A ggplot2 autoplot method for class ScreenMethodResult and SeuratObject

Usage

```
## S3 method for class 'Seurat'
autoplot(
  object,
  reduction = "umap",
  group.by = NULL,
  label = FALSE,
  label.size = 4L,
  pt.size = NULL,
  cols = NULL,
  ...
)

## S3 method for class '`SigBridgeR::ScreenMethodResult`'
autoplot(object, ...)
```

Arguments

object	Seurat object
reduction	Which dimensionality reduction to use. If not specified, first searches for umap, then tsne, then pca
group.by	Name of one or more metadata columns to group (color) cells by (for example, orig.ident); pass 'ident' to group by identity class

label	Whether to label the clusters
label.size	Sets size of labels
pt.size	Adjust point size for plotting
cols	Vector of colors, each color corresponds to an identity class. This may also be a single character or numeric value corresponding to a palette as specified by brewer.pal.info. By default, ggplot2 assigns colors. We also include a number of palettes from the pals package. See DiscretePalette for details.
...	Extra parameters passed to DimPlot

Value

ggplot object

BenchAcc *Benchmark Screening Methods: Classification Accuracy*

Description

Computes binary classification metrics for every screening method present in the input, by comparing each method's per-cell labels against ground-truth labels. "Positive" is treated as the positive class and all other labels as negative, so the accuracy of different screening methods can be compared on the same cells.

Usage

```
BenchAcc(obj, ...)

## S3 method for class 'Seurat'
BenchAcc(
  obj,
  bench_config = character(),
  ...,
  verbose = getFuncOption("verbose"),
  nthreads = getFuncOption("nthreads")
)

## S3 method for class '`SigBridgeR::BenchAccConfig`'
BenchAcc(obj, ..., verbose = getFuncOption("verbose"))

## S3 method for class '`SigBridgeR::ScreenMethodResult`'
BenchAcc(obj, ...)

## Default S3 method:
BenchAcc(obj, ...)
```

Arguments

obj	The object to evaluate, one of: • a Seurat object whose meta.data contains screening method results (columns named after screening methods, e.g. "Scissor"); • a ScreenMethodResult object, whose scRNA_data slot is evaluated; • a BenchAccConfig object, whose screen_method entries are all run through Screen() first and the accumulated labels are evaluated in a single pass.
...	Additional arguments passed between the S3 methods. Currently unused apart from forwarding
bench_config	character. Ground-truth labels used as the reference: either a length-one column name in obj@meta.data, or a per-cell label vector of length ncol(obj). Must contain a "Positive" label. For a BenchAccConfig input the labels are taken from the same-named slot.
verbose	Logical. Whether to report progress information. Default: TRUE.
nthreads	Integer. Number of threads used by the C++ metric computation. Default: 2L.

Details

Dispatch:

BenchAcc() dispatches on the class of obj: a BenchAccConfig is screened first, a ScreenMethodResult is unwrapped to its scRNA_data slot, and BenchAcc.Seurat() performs the computation. Unsupported classes abort with an informative error.

Method detection:

Columns of obj@meta.data whose names match a supported screening method (case-insensitive) are taken as the predicted labels; other columns are ignored. The function aborts when no such column exists.

Ground truth:

A length-one bench_config is looked up as a column of obj@meta.data; otherwise it is used directly as a per-cell label vector. The labels must include "Positive", and the vector form must have one element per cell.

Metrics:

With P, N, TP, FP, TN and FN denoting the numbers of positive cells, negative cells, true/false positives and true/false negatives

Value

A named list with one element per screening method found in meta.data. Each element is a 1-row numeric matrix with columns TPR, FPR, TNR, FNR, F1, Accuracy and so on.

See Also

BenchAccConfig(), ScreenMethodResult, Screen(), BenchGSEA(), BenchMemory(), BenchGeneScore()

Other Benchmark: BenchGSEA(), BenchGeneScore(), BenchMemory()

Examples

```
counts <- matrix(
  rep(1:10, 6L),
  nrow = 10L,
  dimnames = list(paste0("Gene", 1:10), paste0("Cell", 1:6))
)
obj <- Seurat::CreateSeuratObject(counts = counts)
obj$truth <- c("Positive", "Negative", "Positive", "Other", "Positive", "Negative")
obj$Scissor <- c("Positive", "Negative", "Positive", "Other", "Negative", "Negative")
BenchAcc(obj, bench_config = "truth", verbose = FALSE)
```

BenchAccConfig *BenchAccConfig: Accuracy Benchmark Configuration*

Description

BenchAccConfig stores the configuration of an accuracy benchmark run performed by BenchAcc(). It extends BenchConfig with the ground-truth cell labels collected in bench_config; the inputs shared by every screening method (matched_bulk, sc_data, phenotype, label_type, phenotype_class, screen_method) and the method specific arguments collected in method_param are inherited from BenchConfig.

Usage

```
BenchAccConfig(
  matched_bulk,
  sc_data,
  phenotype,
  label_type = "{method_name}",
  phenotype_class = c("binary", "survival", "continuous"),
  screen_method = names(ScreenStrategy),
  method_param = MethodParamSet(),
  bench_config = character(),
  ...
)
```

Arguments

matched_bulk	matrix or data.frame. Bulk expression data (genes x samples) matching phenotype. Shared by all methods.
sc_data	A Seurat object to be screened.
phenotype	Phenotype data, either a named vector or a patient survival data frame. See Screen().
label_type	character or NULL. Length-one glue-style template for the phenotype label type, with {method_name} available. When NULL or not of length one, it falls back to "{method_name}" with an informational message.

phenotype_class
: character. One or more of "binary", "continuous", "survival". Default: all three.

screen_method
: character. One or more method names registered in ScreenStrategy, giving the methods to benchmark and their order. Default: "Scissor".

method_param
: Named list of method-specific arguments, each passed only to the executor of the method it is named after (e.g. list(Scissor_param = list(alpha = 0.05))). Default: MethodParamSet(), i.e. one empty entry per registered method.

bench_config
: character. Ground-truth labels used to score the screening results: either a length-one column name in sc_data@meta.data, or a per-cell label vector of length ncol(sc_data). Must contain a "Positive" label. Default: character(), which is rejected by the validator.

...
: Must be empty. Method specific arguments belong in method_param.

Details

bench_config supplies the reference ("ground-truth") labels that the screening results are scored against. It is a character vector interpreted in one of two ways:

- Length one: the name of a column in sc_data@meta.data holding the per-cell labels. The column must exist and contain a "Positive" level.
- Length greater than one: a per-cell label vector, one label per cell. It must contain a "Positive" level and have the same length as the number of cells in sc_data.

In both cases a "Positive" label is required, since BenchAcc() computes binary classification metrics (TPR, FPR, TNR, FNR, F1, Accuracy) treating "Positive" as the positive class. An empty bench_config is rejected by the validator.

Value

A BenchAccConfig S7 object.

See Also

Other S7-Classes: AnnotationMethod(), BenchConfig(), BenchGSEAConfig(), BenchGeneScoreConfig(), BenchMemoryConfig(), ScreenMethodCache(), ScreenMethodConfig(), ScreenMethodResult()

BenchConfig *BenchConfig: Base Configuration for Benchmarks*

Description

BenchConfig is the common parent of the benchmark configurations in SigBridgeR. It stores the inputs shared by every screening method (matched_bulk, sc_data, phenotype, label_type, phenotype_class, screen_method), the method specific arguments collected in method_param, and an optional param list used by subclasses that forward extra arguments.

Usage

```
BenchConfig(
  matched_bulk,
  sc_data,
  phenotype,
  label_type = "{method_name}",
  phenotype_class = c("binary", "survival", "continuous"),
  screen_method = names(ScreenStrategy),
  method_param = MethodParamSet(),
  sigbridger_version = get_pkg_version(),
  ...
)
```

Arguments

matched_bulk	matrix or data.frame. Bulk expression data (genes x samples) matching phenotype. Shared by all methods.
sc_data	A Seurat object to be screened.
phenotype	Phenotype data, either a named vector or a patient survival data frame. See Screen().
label_type	character or NULL. Length-one glue-style template for the phenotype label type, with {method_name} available. When NULL or not of length one, it falls back to "{method_name}" with an informational message.
phenotype_class	character. One or more of "binary", "continuous", "survival". Default: all three.
screen_method	character. One or more method names registered in ScreenStrategy, giving the methods to benchmark and their order. Default: "Scissor".
method_param	Named list of method-specific arguments, each passed only to the executor of the method it is named after (e.g. list(Scissor_param = list(alpha = 0.05))). Default: MethodParamSet(), i.e. one empty entry per registered method.
sigbridger_version	Package Version of SigBridgeR
...	unused

Details

A single configuration may describe several screening runs: the benchmark screens the same data with each method listed in screen_method, one method at a time.

- screen_method may only contain methods registered in ScreenStrategy (e.g. "Scissor", "scPAS", "LP_SGL"); the constructor of the concrete configuration aborts on any unknown name. It also aborts when the selected methods have no phenotype class in common.
- method_param is a named list with one entry per method, named "<method>_param" (e.g. list(Scissor_param = list(alpha = 0.05))). Each entry is spliced into the arguments of

that method's executor, after its ScreenMethod mapper has run. Use `MethodParamSet()` to build the full set with empty defaults, so unspecified methods simply run with their own defaults.

- `label_type` is a length-one template. The default `"{method_name}"` is resolved by the benchmark with the name of the method currently being benchmarked, so a single configuration can serve methods whose executors expect differently named phenotype labels.

Value

A `BenchConfig` S7 object.

See Also

Other S7-Classes: `AnnotationMethod()`, `BenchAccConfig()`, `BenchGSEAConfig()`, `BenchGeneScoreConfig()`, `BenchMemoryConfig()`, `ScreenMethodCache()`, `ScreenMethodConfig()`, `ScreenMethodResult()`

BenchGeneScore *Single-sample Gene Set Scoring for Single-Cell Data*

Description

Scores one or more gene sets in every cell of a screened single-cell dataset with `ssGSEA`, `AUCell`, `UCell` or `AddModuleScore`. All backends return the same gene-set-by-cell matrix layout, so the cell-level signal recovered by different screening methods can be compared on a common ground.

Usage

```
BenchGeneScore(obj, ...)

## S3 method for class '`SigBridgeR::BenchGeneScoreConfig`'
BenchGeneScore(obj, ..., verbose = getFuncOption("verbose"))

## S3 method for class '`SigBridgeR::ScreenMethodResult`'
BenchGeneScore(obj, ...)

## S3 method for class 'Seurat'
BenchGeneScore(
  obj,
  method = c("ssGSEA", "AUCell", "UCell", "AddModuleScore"),
  ...
)
```

Arguments

`obj`	The object to score, one of: • a `Seurat` object, scored as is; • a ScreenMethodResult object, whose `scRNA_data` slot is scored;

- a BenchGeneScoreConfig object, whose screen_method entries are all run through Screen() first and the accumulated labels are scored in a single pass.

... Additional named arguments forwarded to the selected backend. Arguments are matched against the formal argument names of the backend and split automatically over the two stages when a method needs two calls:

- "ssGSEA": GSVA::ssgseaParam() and GSVA::gsva();
- "AUCell": AUCell::AUCell_buildRankings() and AUCell::AUCell_calcAUC();
- "UCell": UCell::ScoreSignatures_UCell();
- "AddModuleScore": Seurat::AddModuleScore().

verbose Logical. Whether to report progress information. Default: TRUE.

method Character. Scoring backend, one of "ssGSEA", "AUCell", "UCell" or "AddModuleScore". Default: "ssGSEA".

Details

Dispatch:

BenchGeneScore() dispatches on the class of obj, so a BenchGeneScoreConfig is benchmarked and a ScreenMethodResult is unwrapped before the actual scoring happens. BenchGeneScore.Seurat() is itself a generic dispatching on method: every backend lives in its own BenchGeneScoreSeurat.<method> function and its package is loaded on demand by rlang::check_installed(), so neither GSVA, AUCell, UCell nor Seurat is required to merely call the generic.

Config input:

For a BenchGeneScoreConfig input, the configured screen_method entries are run one after another with Screen(). Every run adds its labels to the meta.data of the working Seurat object and the labels are accumulated across methods, after which the scores are computed once. Scores are therefore comparable across screening methods by construction.

Backend notes:

- "UCell" is called with name = "" so that gene-set names stay untouched, and its cell-by-gene-set output is transposed to the common layout.
- "AddModuleScore" drops the genes absent from assay, drops the gene sets that become empty, and stores the scores under an internal .BenchGeneScore prefix so that the meta.data of obj is left untouched. It aborts when no gene set survives the filtering.
- "AUCell" builds the rankings with plotStats = FALSE, since the plot is only meaningful for a manual inspection.

Value

A numeric matrix with gene sets in rows and cells in columns. Gene sets dropped by backend-specific filtering are omitted from the result.

See Also

BenchGeneScoreConfig(), BenchGeneScoreConfigSet(), Screen(), BenchGSEA(), BenchAcc()

Other Benchmark: BenchAcc(), BenchGSEA(), BenchMemory()

Examples

```
counts <- matrix(
  rep(seq_len(20), times = 5) + rep(seq_len(5), each = 20),
  nrow = 20,
  dimnames = list(paste0("Gene", seq_len(20)), paste0("Cell", seq_len(5)))
)
obj <- Seurat::CreateSeuratObject(counts = counts)
obj <- Seurat::NormalizeData(obj, verbose = FALSE)
pathways <- list(
  First = paste0("Gene", 1:5),
  Second = paste0("Gene", 6:10)
)
scores <- try(BenchGeneScore(obj, pathways, method = "ssGSEA", verbose = FALSE)) # require GSVA
```

BenchGeneScoreConfig *BenchGeneScoreConfig: Gene Set Scoring Benchmark Configuration*

Description

BenchGeneScoreConfig stores the configuration of a gene set scoring benchmark run performed by BenchGeneScore(). It extends BenchConfig with the scoring controls collected in bench_config; the inputs shared by every screening method (matched_bulk, sc_data, phenotype, label_type, phenotype_class, screen_method) and the method specific arguments collected in method_param are inherited from BenchConfig.

Usage

```
BenchGeneScoreConfig(
  matched_bulk,
  sc_data,
  phenotype,
  label_type = "{method_name}",
  phenotype_class = c("binary", "survival", "continuous"),
  screen_method = names(ScreenStrategy),
  method_param = MethodParamSet(),
  bench_config = BenchGeneScoreConfigSet(),
  ...
)
```

Arguments

matched_bulk	matrix or data.frame. Bulk expression data (genes x samples) matching phenotype. Shared by all methods.
sc_data	A Seurat object to be screened.
phenotype	Phenotype data, either a named vector or a patient survival data frame. See Screen().

label_type — character or NULL. Length-one glue-style template for the phenotype label type, with {method_name} available. When NULL or not of length one, it falls back to "{method_name}" with an informational message.

phenotype_class — character. One or more of "binary", "continuous", "survival". Default: all three.

screen_method — character. One or more method names registered in ScreenStrategy, giving the methods to benchmark and their order. Default: "Scissor".

method_param — Named list of method-specific arguments, each passed only to the executor of the method it is named after (e.g. list(Scissor_param = list(alpha = 0.05))). Default: MethodParamSet(), i.e. one empty entry per registered method.

bench_config — Named list of gene set scoring controls returned by BenchGeneScoreConfigSet(). It holds method (the scoring backend, one of "ssGSEA", "AUCell" or "UCell") plus any additional arguments passed to that backend. Default: BenchGeneScoreConfigSet(), i.e. ssGSEA.

... — Must be empty. Method specific arguments belong in method_param.

Details

For a BenchGeneScoreConfig input, BenchGeneScore() screens the data with each method in screen_method and scores the accumulated cell labels in a single pass, so the scores are comparable across screening methods by construction. bench_config selects the scoring backend and collects the arguments forwarded to it, as built by BenchGeneScoreConfigSet().

Value

A BenchGeneScoreConfig S7 object.

See Also

Other S7-Classes: AnnotationMethod(), BenchAccConfig(), BenchConfig(), BenchGSEAConfig(), BenchMemoryConfig(), ScreenMethodCache(), ScreenMethodConfig(), ScreenMethodResult()

Examples

```
## Not run:
BenchGeneScoreConfig(
  matched_bulk = bulk_matrix,
  sc_data = seurat,
  phenotype = phenotype,
  label_type = "{method_name}",
  phenotype_class = "binary",
  screen_method = "ssGSEA",
  method_param = list(
    nperm = 1000
  )
)
```

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

`BenchGeneScoreConfigSet` *Build Gene Set Scoring Benchmark Controls*

Description

Collects the arguments of a gene set scoring benchmark into the `bench_config` list of `BenchGeneScoreConfig()`. `method` selects the scoring backend, and additional named arguments are forwarded to that backend when `BenchGeneScore()` computes the scores.

Usage

```
BenchGeneScoreConfigSet(
  method = c("ssGSEA", "AUCell", "UCell", "AddModuleScore"),
  ...
)
```

Arguments

`method`	Character. Scoring backend to benchmark, one of `"ssGSEA"`, `"AUCell"`, `"UCell"`, `"AddModuleScore"`; The package backing the selected backend is checked with `rlang::check_installed()`. Default: `"ssGSEA"`.
`...`	Additional named arguments forwarded to the selected scoring backend. They are matched against the backend's formal arguments and, when a method needs two calls, split over its stages (e.g. `GSVA::ssgseaParam()` and `GSVA::gsva()` for `"ssGSEA"`). Case variants of the function's own arguments are resolved to their canonical names.

Value

A named `list` holding `method` and the extra backend arguments, to be used as the `bench_config` of `BenchGeneScoreConfig()`.

See Also

`BenchGeneScoreConfig()`, `BenchGeneScore()`

Other ConfigSet: `BenchGSEAConfigSet()`, `BenchMemoryConfigSet()`, `MethodParamSet()`

Examples

```
BenchGeneScoreConfigSet(method = "ssGSEA")
```

BenchGSEA *Benchmark Screening Methods: Gene Set Enrichment Analysis*

Description

Runs gene set enrichment analysis (GSEA) on the marker genes of every screening method present in the input, so that the functional signals recovered by different methods can be compared side by side.

Usage

```
BenchGSEA(obj, ...)

## S3 method for class 'Seurat'
BenchGSEA(
  obj,
  group,
  pathway,
  p_value_threshold = NULL,
  ...,
  verbose = getFuncOption("verbose")
)

## S3 method for class '`SigBridgeR::ScreenMethodResult`'
BenchGSEA(obj, ...)

## S3 method for class '`SigBridgeR::BenchGSEAConfig`'
BenchGSEA(obj, ..., verbose = getFuncOption("verbose"))

## Default S3 method:
BenchGSEA(obj, ...)
```

Arguments

obj	A Seurat object whose meta.data contains screening method results (columns named after screening methods, e.g. "Scissor"), a ScreenMethodResult object, or a BenchGSEAConfig object.
...	Additional arguments, automatically split by formal argument names between Seurat::FindMarkers() and fgsea::fgsea() (including fgsea::fgseaMultilevel() / fgsea::fgseaSimple()). ident.1, ident.2 and group.by are reserved by BenchGSEA and must not be supplied.
group	A vector of length equal to the number of cells, with each element being the group label of the corresponding cell.
pathway	A named list of gene sets, each element a character vector of gene symbols. Passed to fgsea::fgsea().

p_value_threshold
: Numeric or NULL. When not NULL, markers are filtered by adjusted p-value (p_val_adj < 0.05) before ranking. Default: NULL.

verbose
: Logical. Whether to report progress. Default: TRUE.

Details

For each screening method found in meta.data:

1. Labels are binarised to "Positive" vs "Other";
2. Seurat::FindMarkers() is run with ident.1 = "Positive", ident.2 = "Other" and group.by set to the method column;
3. Markers are ranked by sign(avg_log2FC) * -log10(p_val_adj);
4. fgsea::fgsea() is run on the ranked list, and the ranking statistics of the leading-edge genes are attached as leadingEdgeStats.

For a BenchGSEAConfig input, every configured screening method is first run via Screen() on the same data before the GSEA comparison.

Value

A data.table of fgsea::fgsea() results stacked across screening methods, with an extra screenMethod column identifying the method and a leadingEdgeStats list column holding the ranking statistics of the leading-edge genes. Methods without markers are dropped with a warning.

See Also

BenchGSEAConfig(), Screen(), fgsea::fgsea(), Seurat::FindMarkers(), BenchAcc(), BenchMemory()

Other Benchmark: BenchAcc(), BenchGeneScore(), BenchMemory()

Examples

```
## Not run:
# sc is a Seurat object already screened by e.g. Scissor and scPAS
pathways <- list(
  pathway_a = c("Gene1", "Gene2", "Gene3"),
  pathway_b = c("Gene4", "Gene5", "Gene6")
)
res <- BenchGSEA(sc, pathway = pathways)

## End(Not run)
```

BenchGSEAConfig *BenchGSEAConfig: GSEA Benchmark Configuration*

Description

BenchGSEAConfig stores the configuration of a GSEA benchmark run performed by BenchGSEA(). It extends BenchConfig with the GSEA-specific controls collected in bench_config; the inputs shared by every screening method (matched_bulk, sc_data, phenotype, label_type, phenotype_class, screen_method) and the method specific arguments collected in method_param are inherited from BenchConfig.

Usage

```
BenchGSEAConfig(
  matched_bulk,
  sc_data,
  phenotype,
  label_type = "{method_name}",
  phenotype_class = c("binary", "survival", "continuous"),
  screen_method = names(ScreenStrategy),
  method_param = MethodParamSet(),
  bench_config = list(),
  ...
)
```

Arguments

matched_bulk	matrix or data.frame. Bulk expression data (genes x samples) matching phenotype. Shared by all methods.
sc_data	A Seurat object to be screened.
phenotype	Phenotype data, either a named vector or a patient survival data frame. See Screen().
label_type	character or NULL. Length-one glue-style template for the phenotype label type, with {method_name} available. When NULL or not of length one, it falls back to "{method_name}" with an informational message.
phenotype_class	character. One or more of "binary", "continuous", "survival". Default: all three.
screen_method	character. One or more method names registered in ScreenStrategy, giving the methods to benchmark and their order. Default: "Scissor".
method_param	Named list of method-specific arguments, each passed only to the executor of the method it is named after (e.g. list(Scissor_param = list(alpha = 0.05))). Default: MethodParamSet(), i.e. one empty entry per registered method.

bench_config	Named list of GSEA controls returned by BenchGSEAConfigSet(). It holds pathway (a named list of gene sets), optionally group (one group label per cell in sc_data), plus optional arguments forwarded to fgsea::fgsea() such as minSize, maxSize and gseaParam. group, when supplied, must have the same length as the number of cells in sc_data. Default: list(), which is rejected by the validator because pathway is required.
...	Must be empty. Method specific arguments belong in method_param.

Value

A BenchGSEAConfig S7 object.

See Also

Other S7-Classes: AnnotationMethod(), BenchAccConfig(), BenchConfig(), BenchGeneScoreConfig(), BenchMemoryConfig(), ScreenMethodCache(), ScreenMethodConfig(), ScreenMethodResult()

BenchGSEAConfigSet *Build GSEA Benchmark Controls*

Description

Collects the arguments of a GSEA benchmark into the bench_config list of BenchGSEAConfig(). Arguments matching fgsea::fgsea() formals are handed over to it; all other named arguments are kept as-is.

Usage

```
BenchGSEAConfigSet(sc_data, pathway, group = NULL, ...)
```

Arguments

sc_data	A Seurat object to be screened. It defines the cells that group assigns labels to.
pathway	A named list of gene sets, each element a character vector of gene symbols. Passed to the pathways argument of fgsea::fgsea().
group	character or NULL. Per-cell group labels, one per cell in sc_data. The vector is stored in bench_config and validated by BenchGSEAConfig(). Default: NULL.
...	Additional named arguments, typically further fgsea::fgsea() controls (e.g. nperm, scoreType). Case variants of the function's own arguments are resolved to their canonical names.

Value

A list of parameters for fgsea::fgsea() and BenchGSEAConfig().

See Also

Other ConfigSet: BenchGeneScoreConfigSet(), BenchMemoryConfigSet(), MethodParamSet()

Examples

```
## Not run:
config <- BenchGSEAConfigSet(
  sc_data = sc,
  pathway = fgsea::examplePathways,
  group = sc$cell_type
)

## End(Not run)
```

`BenchMemory` *Benchmark Screening Methods: Wall Time and Memory*

Description

Runs every screening method listed in a BenchMemoryConfig on the same data and measures its wall time and memory footprint with `bench::mark()`.

Usage

```
BenchMemory(obj, ...)

## S3 method for class '`SigBridgeR::BenchMemoryConfig`'
BenchMemory(obj, ..., verbose = getFuncOption("verbose"))

## Default S3 method:
BenchMemory(obj, ...)
```

Arguments

`obj`	An object of class BenchMemoryConfig
`...`	Additional arguments passed to methods. Currently unused.
`verbose`	`logical`. Whether to report progress. Default: `getFuncOption("verbose")`.

Details

The benchmark is organised in *rounds*. Within a round every method is screened once, and the round order is shuffled by default so that cold-cache, lazy-loading and package-level caching effects are spread across methods instead of always penalising the first one. One row is returned per method per round; summaries across rounds are left to the caller.

Each measurement runs in its own background R process (`callr::r_bg()`) with its own copies of the inputs, staged in a temporary directory with `qs2`. A child process therefore cannot leak package-level caches, loaded Python runtimes or large result objects into the next measurement. The child is always terminated on exit so it cannot outlive the call, and `timeout_sec` bounds the wall time of a single method in a single round.

Three memory numbers are reported because they mean different things:

- mem_alloc - bytes allocated during the run (allocation churn, from bench::mark()).
- mem_peak_mb - peak R heap high-water mark, from gc(reset = TRUE).
- rss_peak_mb - peak process resident set size, recorded only when measure_rss = TRUE (Linux only).

Before the benchmark starts, bench_probe_resources() collects a machine resource summary. Set probe_resources = FALSE in the bench_config controls (see BenchMemoryConfigSet()) to skip this probe, for example when the workspace is large and the probe's full gc() would add noticeable startup time.

Value

A data.table with one row per screening method per round. Each row contains the bench::mark() columns (min, median, itr/sec, mem_alloc, n_itr, n_gc, total_time, ...), with expression replaced by the method name, plus method_name, round, position, status, error, mem_peak_mb, rss_peak_mb and elapsed. Successful measurements have status = "ok"; failed or timed-out measurements are kept with status set to "error" or "timeout" and NA memory metrics, while their benchmark columns are filled with NA.

Benchmark controls (bench_config)

- rounds (3L): number of rounds; every method is screened once per round.
- probe_resources (TRUE): whether to call bench_probe_resources() before staging the inputs. Set to FALSE to skip the probe and its full gc().
- min_iterations, max_iterations (1L, 1L): iteration bounds passed to bench::mark().
- order ("random"): "random" shuffles methods within each round using seed plus the round number; "fixed" keeps the given order.
- seed (getFuncOption("seed")): base seed for shuffling and for each measurement child.
- memory, filter_gc, check, min_time, relative, time_unit: forwarded to bench::mark().
- measure_rss (FALSE): also record peak process RSS (Linux only).
- on_failure ("skip"): "skip" records the failure and continues, "stop" aborts the whole benchmark.
- set_threads (1L): number of threads the child process is pinned to.
- timeout_sec (10800L): maximum wall-clock seconds for one method in one round; use Inf to disable the limit.

See Also

BenchMemoryConfig(), BenchMemoryConfigSet(), bench::mark()

Other Benchmark: BenchAcc(), BenchGSEA(), BenchGeneScore()

Examples

```
## Not run:
config <- BenchMemoryConfig(
  matched_bulk = bulk,
  sc_data = sc,
  phenotype = pheno,
  screen_method = c("scPP", "scPAS", "LP_SGL"),
  method_param = MethodParamSet(list(
    scPP_param = list(Log2FC_cutoff = 0.585),
    scPAS_param = list(permutation_times = 500)
  )),
  bench_config = list(rounds = 3L, timeout_sec = 1800)
)
res <- BenchMemory(config)
res
res[status == "ok", .(median_sec = median(median)), by = method_name]

## End(Not run)
```

BenchMemoryConfig *BenchMemoryConfig: Time and Memory Benchmark Configuration*

Description

BenchMemoryConfig stores the configuration of a time and memory benchmark run performed by BenchMemory(). It extends BenchConfig with the benchmarking controls collected in bench_config; the inputs shared by every screening method (matched_bulk, sc_data, phenotype, label_type, phenotype_class, screen_method) and the method specific arguments collected in method_param are inherited from BenchConfig.

Usage

```
BenchMemoryConfig(
  matched_bulk,
  sc_data,
  phenotype,
  label_type = "{method_name}",
  phenotype_class = c("binary", "survival", "continuous"),
  screen_method = names(ScreenStrategy),
  method_param = MethodParamSet(),
  bench_config = BenchMemoryConfigSet(),
  ...
)
```

Arguments

matched_bulk	matrix or data.frame. Bulk expression data (genes x samples) matching phenotype. Shared by all methods.
sc_data	A Seurat object to be screened.
phenotype	Phenotype data, either a named vector or a patient survival data frame. See Screen().
label_type	character or NULL. Length-one glue-style template for the phenotype label type, with {method_name} available. When NULL or not of length one, it falls back to "{method_name}" with an informational message.
phenotype_class	character. One or more of "binary", "continuous", "survival". Default: all three.
screen_method	character. One or more method names registered in ScreenStrategy, giving the methods to benchmark and their order. Default: "Scissor".
method_param	Named list of method-specific arguments, each passed only to the executor of the method it is named after (e.g. list(Scissor_param = list(alpha = 0.05))). Default: MethodParamSet(), i.e. one empty entry per registered method.
bench_config	Named list of benchmarking controls (rounds, iterations, order, budgets, ...) consumed by BenchMemory(). Default: list(), meaning all defaults. Unknown fields abort.
...	Must be empty. Method specific arguments belong in method_param; unlike the other benchmark configurations nothing is stored in param.

Value

A BenchMemoryConfig S7 object.

See Also

Other S7-Classes: AnnotationMethod(), BenchAccConfig(), BenchConfig(), BenchGSEAConfig(), BenchGeneScoreConfig(), ScreenMethodCache(), ScreenMethodConfig(), ScreenMethodResult()

BenchMemoryConfigSet *Resolve Benchmark Controls*

Description

Merges the user supplied bench_config with the defaults and validates the names, so that a typo aborts instead of being silently ignored.

Usage

```
BenchMemoryConfigSet(
  rounds = 3L,
  probe_resources = TRUE,
  min_iterations = 1L,
  max_iterations = 1L,
  order = c("random", "fixed"),
  seed = getFuncOption("seed"),
  memory = TRUE,
  filter_gc = FALSE,
  check = FALSE,
  measure_rss = FALSE,
  on_failure = c("skip", "stop"),
  set_threads = 1L,
  min_time = 0.5,
  relative = FALSE,
  time_unit = NULL,
  timeout_sec = 10800L,
  ...
)
```

Arguments

rounds	integer. Number of benchmark rounds; every method is screened once per round and per-method summaries are taken across rounds. Odd values let the median ignore a single cold round. Must be at least 1. Default: 3L.
probe_resources	logical. Whether BenchMemory() should call bench_probe_resources() before staging the inputs. The probe calls gc() and can be expensive on large workspaces; disable it when the machine resource summary is not needed. Default: TRUE.
min_iterations	Each expression will be evaluated a minimum of min_iterations times.
max_iterations	Each expression will be evaluated a maximum of max_iterations times.
order	character. Order in which methods run within a round: "random" shuffles them (seeded by seed) to spread cold-cache and package-caching effects; "fixed" keeps the given method order. Default: c("random", "fixed") (first value used).
seed	integer. Base seed for shuffling method order each round; the round number is added to it. Also set inside each measurement child process. Default: 123L.
memory	If TRUE (the default when R is compiled with memory profiling), track memory allocations using utils::Rprofmem(). If FALSE disable memory tracking.
filter_gc	If TRUE remove iterations that contained at least one garbage collection before summarizing. If TRUE but an expression had a garbage collection in every iteration, filtering is disabled, with a warning.
check	Check if results are consistent. If TRUE, checking is done with all.equal(), if FALSE checking is disabled and results are not stored. If check is a function that function will be called with each pair of results to determine consistency.

measure_rss	logical. Whether to record peak process resident set size (RSS) in addition to R heap memory. Supported on Linux only. Default: FALSE.
on_failure	character. Action when a method times out or errors in a round: "skip" warns and continues with the remaining methods, "stop" aborts the whole benchmark. Default: c("skip", "stop") (first value used).
set_threads	integer. Number of threads the measurement child process is pinned to via SetThreads(), so runs stay comparable across methods. Default: 1L.
min_time	The minimum number of seconds to run each expression, set to Inf to always run max_iterations times instead.
relative	If TRUE all summaries are computed relative to the minimum execution time rather than absolute time.
time_unit	If NULL the times are reported in a human readable fashion depending on each value. If one of 'ns', 'us', 'ms', 's', 'm', 'h', 'd', 'w' the time units are instead expressed as nanoseconds, microseconds, milliseconds, seconds, hours, minutes, days or weeks respectively.
timeout_sec	numeric. Maximum wall-clock seconds allowed for one method in one round; the child process is killed when the limit is exceeded. Use Inf to disable the limit. Default: 10800L (3 hours).
...	Expressions to benchmark, if named the expression column will be the name, otherwise it will be the deparsed expression.

Value

A named list of resolved controls.

See Also

Other ConfigSet: BenchGSEAConfigSet(), BenchGeneScoreConfigSet(), MethodParamSet()

Examples

```
BenchMemoryConfigSet()
```

CacheSetHere *Resolve and Set Up Cache Directory*

Description

Resolves a user-specified path to the cache layer directory. In "save" mode, the necessary directory structure is created automatically. In "load" mode, an existing cache directory is identified.

Usage

```
CacheSetHere(
  path,
  cache_config,
  mode = c("load", "save"),
  timestamp = NULL,
  ...
)
```

Arguments

path	character. User-specified path to a cache, root, or parent directory.
cache_config	A ScreenMethodConfig or ScreenMethodCache S7 object. method_name and phenotype_class are extracted from this object to construct the cache directory path.
mode	character. Either "load" or "save". Default: "load".
timestamp	character or NULL. Optional timestamp string for the new cache directory in save mode. When NULL, defaults to format(Sys.time(), "%Y%m%d%H%M%S").
...	Additional arguments (must be empty, checked by rlang::check_dots_empty0()).

Details

The function detects which cache layer the given path belongs to:

- **Cache layer**: The path directly contains a cache_config.json file. In save mode this triggers an error (recursive caching is not supported); in load mode the path is returned as-is.
- **Root layer**: The path's basename matches {method_name}_res (e.g., "Scissor_res"). In save mode a new cache subdirectory is created; in load mode ChooseCache() is called to select an existing cache.
- **Parent layer**: The path is a parent directory. A {method_name}_res subdirectory is created (save) or located (load) underneath it.

In save mode, the cache directory is named {phenotype_class}_{timestamp} (e.g., "survival_20250730120000").

Value

A character string: the absolute path to the cache directory.

See Also

Other cache_config: CacheSysCall(), CheckCache(), ChooseCache(), LoadCache(), WriteCache(), WriteCacheMeta()

Examples

```
## Not run:
config <- ScreenMethodConfig(
  method_name = "Scissor",
  param = list(alpha = 0.05)
)

# Save: create a new cache under the root
CacheSetHere("Scissor_res", config, mode = "save")

# Load: select an existing cache from the root
CacheSetHere("Scissor_res", config, mode = "load")

# Load: point directly to a specific cache
CacheSetHere("Scissor_res/survival_207701011212", config, mode = "load")

## End(Not run)
```

`CacheSysCall` *System-Level Cache Load and Save*

Description

High-level wrapper for loading and saving cached screening results. Combines cache path resolution (via `CacheSetHere()`), configuration validation (via `CheckCache()`), and data I/O (via `LoadCache()` or `WriteCache()`) into a single call.

Usage

```
CacheSysCall(
  mode = c("load", "save"),
  path = NULL,
  cache,
  verbose = TRUE,
  timestamp = NULL,
  ...
)
```

Arguments

`mode`	`character`. Either `"load"` or `"save"`.
`path`	`character` or `NULL`. Path passed to `CacheSetHere()`. In save mode a `NULL` value falls back to `cache@cache_path`. In load mode it is mandatory: a `NULL` value aborts, no cache is looked up implicitly. Default: `NULL`.
`cache`	A `ScreenMethodConfig` or `ScreenMethodCache` S7 object. For save mode, a `ScreenMethodCache` is required (contains the data to write). For load mode, either class is accepted.

verbose	logical. Whether to print progress messages. Default: TRUE.
timestamp	character or NULL. Optional timestamp string for the new cache directory in save mode. Passed to CacheSetHere().
...	Additional arguments passed to WriteCache() in save mode (e.g., format, max_rows, max_cols).

Details

CacheSysCall is the primary entry point for cache operations in the SigBridgeR pipeline. It dispatches to either CacheSysCall.load() or CacheSysCall.save() based on the mode argument.

Load mode:

In "load" mode, the function:

1. Resolves the cache path via CacheSetHere().
2. Validates cache consistency via CheckCache().
3. Loads each cached data object via LoadCache() and returns them as a named list.

Save mode:

In "save" mode, the function:

1. Resolves and creates the cache directory via CacheSetHere().
2. Writes the cache data and metadata via WriteCache().
3. Returns the cache data invisibly.

Value

In "load" mode, a named list of loaded cache data objects. In "save" mode, the cache data invisibly.

See Also

Other cache_config: CacheSetHere(), CheckCache(), ChooseCache(), LoadCache(), WriteCache(), WriteCacheMeta()

Examples

```
## Not run:
# Load cached results
cache <- ScreenMethodCache(
  cache_path = "Scissor_res/",
  cache_config_path = "Scissor_res/cache_config.json",
  cache_data = list(),
  screen_method_config = ScreenMethodConfig(
    screen_method = "Scissor",
    param = list(alpha = 0.05)
  )
)
result <- CacheSysCall(
  "load",
  path = "Scissor_res/survival_2025_01_01",
```

```
  cache = cache
)

# Save results to cache
cache <- ScreenMethodCache(
  cache_path = "Scissor_res/",
  cache_config_path = "Scissor_res/cache_config.json",
  cache_data = list(result = my_matrix),
  screen_method_config = ScreenMethodConfig(
    screen_method = "Scissor",
    param = list(alpha = 0.05)
  )
)
CacheSysCall("save", cache = cache)

## End(Not run)
```

`CellTypistAnnotate` *Annotate Cell Types Using CellTypist (Python Backend)*

Description

Performs automated cell type annotation via the `celltypist` Python package using `reticulate` integration. Accepts a `Seurat` object as input and returns it enriched with CellTypist prediction results as metadata columns.

Requires a Python environment with `celltypist` installed (see https://github.com/Teichlab/celltypist). By default the function locates a suitable Python interpreter automatically via `reticulate`; use `reticulate::use_python()` to pin a specific environment beforehand.

Usage

```
CellTypistAnnotate(sc, ...)

## Default S3 method:
CellTypistAnnotate(sc, ...)

## S3 method for class '`SigBridgeR::ScreenMethodResult`'
CellTypistAnnotate(sc, ...)

## S3 method for class 'SingleCellExperiment'
CellTypistAnnotate(sc, ...)

## S3 method for class 'Seurat'
CellTypistAnnotate(
  sc,
  model = NULL,
  download = TRUE,
  force_update = FALSE,
```

```
  majority_voting = FALSE,
  mode = c("best match", "prob match"),
  p_thres = 0.5,
  transpose_input = FALSE,
  gene_file = NULL,
  cell_file = NULL,
  over_clustering = NULL,
  use_GPU = FALSE,
  min_prop = 0L,
  verbose = getFuncOption("verbose"),
  assay = "RNA",
  ...
)
```

Arguments

sc	A single-cell dataset, one of: • A Seurat object (recommended) with a valid assay and gene expression matrix; • A SingleCellExperiment object, converted to Seurat via Seurat::as.Seurat() before annotation (requires counts/logcounts assays); • A ScreenMethodResult object, whose scRNA_data slot is annotated.
...	Additional arguments passed through to the CellTypist Python backend.
model	Character. CellTypist model specification. One of: • A built-in model name (e.g., "Immune_All_Low"); a .pkl suffix is appended automatically if missing. • A path to a local .pkl model file.
download	Logical. Whether to automatically download the model first. Default: TRUE.
force_update	Logical. Whether to force re-downloading the model files when download = TRUE. Default: FALSE.
majority_voting	Logical. Whether to refine predicted labels by running a majority voting classifier after over-clustering. Default: FALSE.
mode	Character. Prediction mode: "best match" selects the cell type with the largest score; "prob match" enables multi-label classification. Default: "best match".
p_thres	Numeric. Probability threshold for multi-label classification in "prob match" mode. Ignored when mode = "best match". Default: 0.5.
transpose_input	Logical. Whether to transpose the input matrix. Set to TRUE if the input is in gene-by-cell format. Default: FALSE.
gene_file	Character. Path to a file with genes (one per line) corresponding to rows of the input matrix. Default: NULL.
cell_file	Character. Path to a file with cells (one per line) corresponding to columns of the input matrix. Default: NULL.

over_clustering
: Character or vector. Over-clustering specification used for majority voting: (1) path to a plain file with one cluster ID per line; (2) metadata column name in AnnData; (3) vector/array of cluster assignments; or (4) omitted for the heuristic approach. Ignored when `majority_voting = FALSE`. Default: `NULL`.

use_GPU
: Logical. Whether to use GPU acceleration (via rapids-singlecell) for over-clustering. Only relevant when `majority_voting = TRUE`. Default: `FALSE`.

min_prop
: Numeric. Minimum proportion of the dominant cell type required to name a subcluster; subclusters below the threshold are labeled `'Heterogeneous'`. Ignored when `majority_voting = FALSE`. Default: `0L`.

verbose
: Logical. Whether to print progress messages during annotation. Default: `getFuncOption("verbose")`.

assay
: which Seurat assay to use

Value

The input `Seurat` object with prediction results added: cell type labels are stored as new columns in `meta.data` (prefixed with `celltypist_`), and the scoring matrices are stored in `misc$celltypist`. Column names may vary slightly depending on the CellTypist version and options used. `SingleCellExperiment` and ScreenMethodResult inputs are annotated as `Seurat` objects and returned as such.

Requirements

- R packages: `reticulate`, `AnnDataR`
- Python packages: `celltypist`, `scanpy`, `anndata`
- A working Python environment discoverable by `reticulate`

See Also

Other Single_Cell_Annotation_Method: `DownloadCellTypistModels()`, `ListCellTypistModels()`, `RegisterAnnoMethod()`, `SCAnnotateStrategy`, `SingleRAnnotate()`, `mLLMCelltypeAnnotate()`

Examples

```
## Not run:
# Use a built-in immune model with majority voting
annotated <- CellTypistAnnotate(
  sc = seurat_obj,
  model = "Immune_All_Low",
  majority_voting = TRUE
)

# Probability-based multi-label annotation
annotated <- CellTypistAnnotate(
  sc = seurat_obj,
  model = "Immune_All_Low",
  mode = "prob match",
  p_thres = 0.5
)
```

```
# Use a local model file
annotated <- CellTypistAnnotate(
  sc = seurat_obj,
  model = "/path/to/My_Model.pkl"
)

## End(Not run)
```

CheckCache *Check Cache Configuration Consistency*

Description

Validates whether the cached configuration matches the current parameters. This function reads a cache metadata JSON file and compares it with the provided screen method, phenotype class, label type, and parameters.

Usage

```
CheckCache(cache_config, ...)
```

Arguments

cache_config	A ScreenMethodConfig or ScreenMethodCache S7 object containing the current configuration to validate.
...	Additional arguments to be passed to the method: For ScreenMethodConfig: • path: character. Path to the cache directory or directly to the cache_config.json file. Required for ScreenMethodConfig

Details

The function compares each field in the current configuration against the cached metadata. If any field differs (method name, method version, phenotype class, label type, or parameters), the function aborts with a formatted error message showing which parameter differs and both the provided and actual values.

Value

Invisible TRUE if the cache configuration is consistent with the current parameters. Otherwise, aborts with an error message displaying the differences.

Methods

CheckCache is an S7 generic with methods available for the following classes:

- ANY
- SigBridgeR::ScreenMethodCache
- SigBridgeR::ScreenMethodConfig

See Also

Other cache_config: CacheSetHere(), CacheSysCall(), ChooseCache(), LoadCache(), WriteCache(), WriteCacheMeta()

Examples

```
## Not run:
# Check cache consistency for a ScreenMethodConfig
config <- ScreenMethodConfig(
  method_name = "Scissor",
  param = list(alpha = 0.05)
)
CheckCache(config, path = "cache/cache_config.json")

# Check using a ScreenMethodCache object
cache <- ScreenMethodCache(
  cache_path = "cache/",
  cache_config_path = "cache/cache_config.json",
  cache_data = list(),
  screen_method_config = config
)
CheckCache(cache)

## End(Not run)
```

CheckNA *Check for Missing Values (NA) in Data*

Description

Identifies and reports the presence of missing values (NA) in vectors or two-dimensional data structures. Provides detailed positional information including row/column indices and names when available. Returns structured information for programmatic use while displaying user-friendly messages.

Usage

```
CheckNA(data, max_print = 5L, ...)
```

Arguments

data	A vector, matrix, data frame, or Matrix object to check for NA values.
max_print	Integer. Maximum number of NA positions to display in console output. Set to 0 to suppress position details. Default: 5L.
...	Additional arguments (currently unused, reserved for future extensions).

Value

Invisible list containing NA information:

- `count`: Total number of NA values found
- `positions`: For vectors: integer vector of positions; For 2D data: data frame with row/column indices and names (if available)
- `names`: For named vectors: character vector of names corresponding to NA positions

Returns empty list components when no NA values are found.

Output Behavior

- **No NAs found**: Prints success message and returns empty list
- **NAs found**: Prints warning with count and up to `max_print` positions
- **Position details**: Includes row/column names when available for easier debugging

See Also

`is.na`, `complete.cases`, `na.omit`

Examples

```
## Not run:
# Example 1: Vector with NAs
vec <- c(10, 20, NA, 40, NA, 60)
result <- CheckNA(vec)
# Output: Found 2 NA values
#         Positions: 3, 5

# Example 2: Named vector
named_vec <- c(Sample1 = 10, Sample2 = NA, Sample3 = 30, Sample4 = NA)
result <- CheckNA(named_vec)
# Output: Found 2 NA values
#         Positions: 2, 4
#         with names: Sample2, Sample4

# Example 3: Data frame with row and column names
df <- data.frame(
  Gene1 = c(1.0, 2.0, NA),
  Gene2 = c(NA, 2.5, 3.0),
  Gene3 = c(1.5, NA, 3.5),
  row.names = c("Cell1", "Cell2", "Cell3")
)
result <- CheckNA(df)
# Output: Found 3 NA values in data
#         First 3 positions:
#         Row 3 (Cell3), Col 1 (Gene1)
#         Row 1 (Cell1), Col 2 (Gene2)
#         Row 2 (Cell2), Col 3 (Gene3)
```

```
# Example 4: Matrix without names
mat <- matrix(c(1, NA, 3, 4, 5, NA), nrow = 2, ncol = 3)
result <- CheckNA(mat)
# Output: Found 2 NA values in data
#         First 2 positions:
#         Row 2, Col 1
#         Row 1, Col 3

# Example 5: No NAs found
clean_data <- c(1, 2, 3, 4, 5)
result <- CheckNA(clean_data)
# Output: No NA values found in the vector

# Example 6: Programmatic use - extract NA count
df <- data.frame(a = c(1, NA, 3), b = c(NA, 2, 3))
na_info <- CheckNA(df)
na_info$count  # Returns: 2
na_info$positions  # Returns: data frame with positions

## End(Not run)
```

ChooseCache *Choose a Cache Directory*

Description

Selects a cache directory from the specified path. If only one cache subdirectory exists, it is returned automatically; if multiple are found, an interactive menu is presented for the user to choose.

Usage

```
ChooseCache(directory)
```

Arguments

directory	character. Path to the parent directory containing cache subdirectories. The directory name must match {method_name}_res for a registered screening method.

Details

The function expects the parent directory name to follow the convention {method_name}_res (e.g., "Scissor_res", "scPAS_res"). It validates the directory name against the registered screening methods in ScreenStrategy.

If no cache subdirectories are found, the function aborts with an error. If exactly one subdirectory exists, it is returned without prompting. If multiple subdirectories exist, utils::menu() is used to present an interactive selection menu.

Value

A `character` string: the path to the selected cache directory.

See Also

Other cache_config: `CacheSetHere()`, `CacheSysCall()`, `CheckCache()`, `LoadCache()`, `WriteCache()`, `WriteCacheMeta()`

Examples

```
## Not run:
cache_path <- ChooseCache("./Scissor_res")

## End(Not run)
```

ChooseNormalization *Data-Driven Selection of Single-Cell Normalization Methods*

Description

A quantitative framework for selecting optimal normalization strategies (e.g., SCTransform vs. LogNormalization) based on diagnostic metrics rather than heuristics. Evaluates three critical dimensions of preprocessing quality:

1. **Variance stabilization**: Decoupling of mean-variance relationship in normalized expression (lower correlation = better).
2. **Biological signal retention**: Preservation of known marker genes within highly variable genes (higher retention = better).
3. **Dropout robustness**: Removal of technical dropout bias from normalized values (lower correlation with dropout rate = better).

Methods are ranked using a weighted composite score. Designed for head-to-head comparison of preprocessed Seurat objects.

Usage

```
ChooseNormalization(
  ...,
  subset_size = integer(),
  known_hvgs = list(),
  n_hvgs = 2000L,
  low_expressed_thresh = 0.2,
 weight = c(variance_stability = 0.4, marker_signal = 0.35, dropout_robustness = 0.25),
  verbose = getFuncOption("verbose"),
  seed = getFuncOption("seed")
)
```

Arguments

...	Named arguments where each value is a Seurat object representing a distinct preprocessing strategy (e.g., SCT = sct_obj, Log = log_obj). **Requirements**: • Must be named (names become method identifiers) • Must contain normalized data in the data slot of the specified assay/layer • Must have identical cell counts (for fair comparison)
subset_size	Integer. Number of cells to subsample for diagnostics. If missing or empty, defaults to min(10000, total_cells). Smaller subsets accelerate computation with minimal accuracy loss for large datasets (>50k cells).
known_hvgs	Named list of canonical marker genes for cell types. Format: list(cell_type1 = c("geneA", "geneB"), cell_type2 = c("geneC", ...)). Used to compute marker_retention metric. If NULL (default), this metric is omitted from scoring.
n_hvgs	Integer. Number of top highly variable genes to evaluate for marker retention. Default: 2000L. Only relevant when known_hvgs is provided.
low_expressed_thresh	Numeric (0-1). Quantile threshold for filtering lowly expressed genes during variance-mean correlation calculation. Genes below this quantile of mean expression are excluded. Default: 0.2 (bottom 20% excluded).
weight	Named numeric vector specifying weights for composite scoring. Must contain exactly these components summing to 1: • variance_stability: Weight for variance-mean decoupling (default: 0.4) • marker_signal: Weight for marker gene retention (default: 0.35) • dropout_robustness: Weight for dropout bias removal (default: 0.25)
verbose	Logical. Whether to print progress messages.
seed	Integer. Seed for reproducible random sampling.

Value

A list containing:

- metrics: A data.table with diagnostic metrics per method:
  - variance_mean_cor: Pearson correlation between log10(mean) and log10(variance) of normalized expression (lower = better)
  - marker_retention: Proportion of known markers in top HVGs (higher = better; NA if known_hvgs not provided)
  - mean_dropout_residual: Absolute Spearman correlation between dropout rate (from counts) and normalized means (lower = better)
  - composite_score: Weighted combination of normalized metrics (0-1 scale)
  - rank: Method ranking (1 = best)
- recommendation: Character string naming the top-ranked method
- plots: (Optional) Diagnostic visualizations if ggplot2 available

Workflow

1. Validates input objects (naming, cell count consistency, data slot presence)
2. Subsamples cells (if needed) for computational efficiency
3. Computes three core metrics per method:
   - **Variance stabilization**: Correlation between log-transformed mean and variance of normalized expression
   - **Marker retention**: Overlap between user-provided markers and top HVGs
   - **Dropout robustness**: Correlation between gene dropout rate (from counts) and normalized expression means
4. Normalizes metrics to 0-1 scale (inverting where lower=better)
5. Computes weighted composite score and ranks methods
6. Returns top recommendation with full diagnostic report

Notes

- Requires *pre-normalized* Seurat objects-this function **does not** perform normalization itself. Users must first run `SCTransform()`, `NormalizeData()`, etc., and store results in the `data` slot.
- Marker retention metric is only computed when `known_hvgs` is provided. Without it, scoring relies solely on variance stabilization and dropout robustness.
- Weights should sum to 1 (not enforced but recommended for interpretable scores).

Examples

```
## Not run:
# Compare two preprocessed Seurat objects
sct_obj <- SCTransform(seurat_raw, verbose = FALSE)
log_obj <- NormalizeData(seurat_raw, normalization.method = "LogNormalize")

result <- ChooseNormalization(
  SCT = sct_obj,
  Log = log_obj,
  subset_size = 5000,
  verbose = TRUE
)

# Top recommendation
result$recommendation

# Full metrics table
result$metrics[, .(method, composite_score, rank)]

# With marker genes for refined scoring
markers <- list(
  T_cell = c("CD3D", "CD3E", "CD8A"),
  B_cell = c("CD79A", "MS4A1", "CD19"),
  Myeloid = c("CD14", "LYZ", "FCGR3A")
)
```

```
result <- ChooseNormalization(
  SCT = sct_obj,
  Log = log_obj,
  known_hvgs = markers,
  weight = c(variance_stability = 0.3, marker_signal = 0.5, dropout_robustness = 0.2)
)

## End(Not run)
```

CountsToTPM *Convert bulk RNA-seq raw counts to TPM*

Description

Convert bulk RNA-seq raw counts to TPM

Usage

```
CountsToTPM(counts, gene_length)
```

Arguments

counts	A base matrix or Matrix object. Rows are genes, columns are samples.
gene_length	A named numeric vector or named list. Names are gene IDs, values are gene lengths in bp.

Value

TPM matrix. Base matrix input returns base matrix; Matrix input returns Matrix object.

Examples

```
## Not run:
# Create a counts matrix (3 genes x 2 samples)
counts <- matrix(
  c(100, 200, 300, 150, 250, 350),
  nrow = 3, ncol = 2,
  dimnames = list(c("GENE1", "GENE2", "GENE3"), c("Sample1", "Sample2"))
)

# Gene lengths in bp (must be named and match rownames)
gene_length <- c(GENE1 = 1000, GENE2 = 2000, GENE3 = 1500)

# Convert counts to TPM
tpm <- CountsToTPM(counts, gene_length)
colSums(tpm)  # Each column should sum to 1e6

# Also works with sparse matrices (dgCMatrix)
```

```
sparse_counts <- Matrix::Matrix(counts, sparse = TRUE)
tpm_sparse <- CountsToTPM(sparse_counts, gene_length)

## End(Not run)
```

DoDEGAS *Run DEGAS Analysis for Single-Cell and Bulk RNA-seq Data Integration*

Description

This function performs DEGAS to integrate single-cell and bulk RNA-seq data, identifying phenotype-associated cells using a bootstrap aggregated multi-task learning approach.

Usage

```
DoDEGAS(
  select_fraction = 0.05,
  min_thresh = 0.4,
  matched_bulk,
  sc_data,
  phenotype = NULL,
  sc_data.pheno_colname = NULL,
  label_type = "DEGAS",
  phenotype_class = c("binary", "continuous", "survival"),
  tmp_dir = "DEGAS_res",
  env_params = lifecycle::deprecated(),
  degas_params = list(),
  normality_test_method = "jarque-bera",
  ...
)
```

Arguments

select_fraction	The top percentage of selected cells will be considered as Positive cells, without considering how much larger the possible correlation coefficient of the observation group is compared to that of the control group. Only usedl when phenotype_class is "binary" or "survival". (default: 0.05)
min_thresh	DEGAS will calculate the possible correlation coefficients for each cell related to the phenotype. When the coefficient of the observation group is at least min_thresh larger than that of the control group, it can be considered related to the phenotype and will be marked as Positive. The priority of min_thresh is higher than that of select_fraction. (default: 0.4)
matched_bulk	Bulk RNA-seq data as matrix or data.frame (rows=genes, columns=samples)
sc_data	Single-cell data as Seurat object containing RNA assay

phenotype — Bulk-level phenotype data. For classification: binary matrix with one-hot encoding. For survival: matrix with two columns (time and event status). Can be NULL, matrix, data.frame, or vector.

sc_data.pheno_colname — Column name for single-cell phenotype in metadata (if available), default: NULL

label_type — Label type for DEGAS results (default: "DEGAS")

phenotype_class — Type of phenotype: "binary" (classification), "continuous", or "survival"

tmp_dir — (Soft-deprecated) Temporary directory for DEGAS intermediate files (default: "DEGAS_res"). Acts as fallback for load_cache and save_cache when not specified via .... Prefer using load_cache / save_cache in ... instead. See CacheSetHere().

env_params — **[Deprecated]** List of environment parameters for Python setup including:

- env.name: environment name (default: "r-reticulate-degas")
- env.type: environment type "conda", "environment", or "venv" (default: "environment")
- env.method: environment setup method "system", "conda" (default: "system")
- env.file: path to environment file (default: system.file("conda/DEGAS_environment.yml", package = "SigBridgeR"))
- env.python_version: Python version (default: "3.9.15")
- env.packages: named vector of Python packages and versions (default: c("tensorflow" = "2.4.1", "protobuf" = "3.20" ,"numpy" = "any"))
- env.recreate: whether to recreate environment (default: FALSE)
- env.use_conda_forge: whether to use conda-forge channel (conda only, default: TRUE)
- env.verbose: verbose output (default: FALSE)

degas_params — List of DEGAS algorithm parameters including:

- DEGAS.model_type: model type ("BlankClass", "ClassBlank", "ClassClass", "ClassCox", "BlankCox")
- DEGAS.architecture: "Standard" (feed forward) or "DenseNet" (dense net), default: "DenseNet"
- DEGAS.ff_depth: number of layers in model (>=1, default: 3)
- DEGAS.pyloc: path to Python executable (default: NULL, automatic detection)
- DEGAS.bag_depth: bootstrap aggregation depth (>=1, default: 5)
- DEGAS.train_steps: training steps (default: 2000)
- DEGAS.scbatch_sz: single-cell batch size (default: 200)
- DEGAS.patbatch_sz: patient batch size (default: 50)
- DEGAS.hidden_feats: hidden features (default: 50)
- DEGAS.do_prc: dropout percentage (default: 0.5)
- DEGAS.lambda1: regularization parameter 1 (default: 3.0)
- DEGAS.lambda2: regularization parameter 2 (default: 3.0)
- DEGAS.lambda3: regularization parameter 3 (default: 3.0)

- DEGAS.seed: random seed (default: 2)

`normality_test_method`
: Method for normality testing: "jarque-bera", "d'agostino", or "kolmogorov-smirnov"

`...`
: Additional arguments. Currently supports:
  - `verbose`: Logical indicating whether to print progress messages. Defaults to `TRUE`.
  - `assay`: Name of assay to use. Defaults to "RNA".
  - `load_cache`: Cache directory path for loading a precomputed DEGAS model. Supports root-level, cache-level, or parent-level paths. See `CacheSetHere()`.
  - `save_cache`: Cache directory path for saving the trained DEGAS model. Supports root-level or parent-level paths. See `CacheSetHere()`.

Details

The function performs the following steps:

1. Validates input data and parameters
2. Sets up Python environment with required dependencies
3. Trains bootstrap aggregated DEGAS model using `runCCMTLBag`
4. Generates cell-level predictions using `predClassBag`
5. Applies statistical testing to identify phenotype-associated cells
6. Labels cells as "Positive" or "Other" based on selection criteria

Model type is automatically determined:

- BlankClass: only bulk phenotype specified (scLab = NULL)
- ClassBlank: only single-cell phenotype specified (patLab = NULL)
- ClassClass: both single-cell and bulk phenotypes specified
- ClassCox: single-cell phenotype + bulk survival data
- BlankCox: only bulk survival data specified

Value

A list containing:

- scRNA_data: Seurat object with DEGAS labels added to metadata
- model: The model trained using the input data, andit can be used for cell classification prediction.
- DEGAS_prediction: Data table with DEGAS predictions containing:
  - Predicted label probabilities for each cell
  - Cell labels ("Positive"/"Other") based on selection criteria
  - Difference scores for binary phenotypes
  - Cell identifiers

References

Johnson TS, Yu CY, Huang Z, Xu S, Wang T, Dong C, et al. Diagnostic Evidence GAuge of Single cells (DEGAS): a flexible deep transfer learning framework for prioritizing cells in relation to disease. Genome Med. 2022 Feb 1;14(1):11.

See Also

Other screen_method: DoLP_SGL(), DoPIPET(), DoPhenoMapR(), DoSCIPAC(), DoSIDISH(), DoScissor(), DoTiRank(), DoscAB(), DoscPAS(), DoscPP()

Other DEGAS: DEGASEnvSet(), DEGASModelDetect(), DEGASParamSet(), TrainDEGASModel(), ValidateDEGASParams()

Examples

```
## Not run:
# Binary classification example
result <- DoDEGAS(
  select_fraction = 0.05, # `select_fraction` only used in binary and survival phenotyping
  matched_bulk = bulk_matrix,
  sc_data = seurat_obj,
  phenotype = bulk_phenotype,
  phenotype_class = "binary"
)

# Survival analysis example
result <- DoDEGAS(
  select_fraction = 0.05, # `select_fraction` only used in binary and survival phenotyping
  matched_bulk = bulk_matrix,
  sc_data = seurat_obj,
  phenotype = survival_data,
  phenotype_class = "survival"
)

## End(Not run)
```

DoDEGASv2 *Screen Function Template*

Description

Screen Function Template

Usage

```
DoDEGASv2(
  matched_bulk,
  sc_data,
```

```
  phenotype,
  label_type = NULL,
  phenotype_class = c("binary", "survival", "continuous"),
  top_fraction_pos = 0.2,
  sclab = NULL,
  bulk_hvg = TRUE,
  bulk_de = TRUE,
  sc_de = TRUE,
  add_genes = NULL,
  n_hvg = 250L,
  n_bulk_de = 250L,
  n_sc_de = 200L,
  padj.thresh = 0.05,
  only.pos = FALSE,
  min.pct = 0.25,
  logfc.threshold = 0.25,
  n_st_classes = length(unique(sc_data$seurat_clusters)),
  loss_type = c("cross_entropy", "log_neg", "rank_loss"),
  transfer_type = c("Wasserstein", "MMD"),
  model_save_dir = "DEGASv2_res",
  lambda1 = 1L,
  lambda2 = 3L,
  lambda3 = 3L,
  tot_seeds = 10L,
  tot_iters = 300L,
  extract_embs = FALSE,
  random_feat = FALSE,
  random_perc = 0.8,
  early_stopping = FALSE,
  ...
)
```

Arguments

`matched_bulk`	Matrix or data frame of preprocessed bulk RNA-seq expression data (genes x samples). Column names must match names/IDs in `phenotype`.
`sc_data`	A matrix/Matrix (genes x cells) or a Seurat object containing scRNA-seq data to be screened.
`phenotype`	Phenotype data, either: • Named vector (names match `matched_bulk` columns), or • Patient survival Data frame with row names matching `matched_bulk` columns, colnames named "time" and "status"
`label_type`	Character specifying phenotype label type
`phenotype_class`	Type of phenotypic outcome (must be consistent with input data): • `"binary"`: Binary traits (e.g., case/control) • `"continuous"`: Continuous measurements

- "survival": Survival infomation

top_fraction_pos
: Proportion of cells to be labelled as "Positive"

sclab
: Metadata for scRNA-seq e.g. cell type, clusters (required), default = NULL

bulk_hvg
: Whether user wants to use sample (bulk) level high variable genes as input in DEGAS. Default is TRUE.

bulk_de
: Whether user wants to use sample (bulk) differential expression genes for phenotype using DEseq2, commonly set Normal or Control as reference. Default is TRUE.

sc_de
: Whether user wants to use single cell differential expression genes for each cluster. Default is TRUE.

add_genes
: The user decides whether to add their own gene list by preference. If the user set bulk_hvg, bulk_de and sc_de as FALSE, this gene list is necessary. Default is NULL.

n_hvg
: Number of bulk high variable genes, default = 250.

n_bulk_de
: Number of bulk differential expression genes for phenotype, default = 250.

n_sc_de
: Number of genes for each single cell clusters, default = 200.

padj.thresh
: Threshold for differential expression analysis, default = 0.05

only.pos
: Only return positive markers (FALSE by default)

min.pct
: only test genes that are detected in a minimum fraction of min.pct cells in either of the two populations. Meant to speed up the function by not testing genes that are very infrequently expressed. Default is 0.01

logfc.threshold
: Limit testing to genes which show, on average, at least X-fold difference (log-scale) between the two groups of cells. Default is 0.1 Increasing logfc.threshold speeds up the function, but can miss weaker signals. If the slot parameter is "scale.data" no filtering is performed.

n_st_classes
: Integer specifying the number of distinct single-cell/spatial labels (cell types, clusters, or spatial annotations). This defines the output dimension of the high-resolution (single-cell/spatial) branch, and is passed to the Python backend as high_reso_output_shape. It is required for Class* model types, which use single-cell labels to supervise the high-resolution branch; for Blank* models it is an unused dummy value. Typically set to length(unique(labels)). Default = NULL.

loss_type
: String specifying the loss function used for training the low-resolution (patient) branch. Typically, classification models use cross_entropy, while survival models can use log_neg or rank_loss. This value must match the expected loss type of the chosen model_type, otherwise the backend will raise an error.

transfer_type
: String indicating the cross-domain alignment method. The backend currently supports Wasserstein (adversarial with gradient penalty) and MMD (maximum mean discrepancy). This parameter determines how the transfer alignment loss is computed.

model_save_dir
: String specifying the output directory for model files. The function will automatically create the directory if it does not exist and will save configuration files, prediction results, and optionally embedding vectors for each submodel.

lambda1	Numeric value, a weighting coefficient that controls the contribution of the high-resolution branch (single-cell/spatial supervision loss). Only applicable to models with the Class prefix. Default = 1.
lambda2	Numeric value, a weighting coefficient that controls the loss strength for the low-resolution (patient) branch. This parameter applies to all model types. Default = 2.
lambda3	Numeric value, a weighting coefficient that controls the contribution of the domain-alignment (transfer) loss. Works jointly with transfer_type to regulate the alignment between high- and low-resolution embedding distributions. Default = 3.
tot_seeds	Integer specifying the number of bagging iterations. Default = 10.
tot_iters	Integer specifying the total number of training iterations (epochs).
extract_embs	Logical value. If TRUE, saves the high- and low-resolution hidden embeddings as .npy files at each save interval for downstream visualization or analysis. Default = FALSE.
random_feat	Logical value. If TRUE, randomly selects a subset of gene features before each submodel training. Default = FALSE.
random_perc	Numeric value between 0 and 1 specifying the proportion of features to retain when random_feat is enabled default = 0.8. The same proportion is applied to both patient-level and single-cell/spatial expression matrices.
early_stopping	Logical value. A reserved flag for early stopping;
...	Additional arguments passed to the function. Common parameters include: • verbose: Logical. Whether to print verbose output (default: TRUE). • seed: Integer. Random seed for reproducibility (default: 123). • Other parameters are passed to Seurat::FindAllMarkers()

Value

A named list containing:

scRNA_data Modified single-cell data object with integrated screening results.

DoLP_SGL *Perform LP-SGL Screening Analysis*

Description

Identifies phenotype-associated cell subpopulations using Lasso-Penalized Sparse Group Lasso (LP-SGL) with Leiden community detection. This method integrates bulk and single-cell RNA-seq data to identify cell subpopulations associated with phenotypic outcomes.

Usage

```
DoLP_SGL(
  matched_bulk,
  sc_data,
  phenotype,
  label_type = "LP_SGL",
  phenotype_class = c("binary", "continuous", "survival"),
  family = lifecycle::deprecated(),
  resolution = 0.6,
  alpha = 0.5,
  nfold = 5L,
  dge_analysis = list(run = FALSE, logFC_threshold = 1L, pval_threshold = 0.05),
  ...
)
```

Arguments

matched_bulk	Matrix or data frame of preprocessed bulk RNA-seq expression data (genes x samples). Column names must match names/IDs in phenotype.
sc_data	A Seurat object containing scRNA-seq data to be screened.
phenotype	Phenotype data, either: • Named vector, where names match columns of matched_bulk • Patient survival data frame, where row names match columns of matched_bulk, and column names are "time" and "status"
label_type	Character specifying phenotype label type, e.g. "SBS1" or "time".
phenotype_class	Type of phenotypic outcome. Must be consistent with input data: • "binary": Binary traits, e.g. case/control • "continuous": Continuous measurements • "survival": Survival information
family	**[Deprecated]** Type of regression model: "logit" (logistic), "cox" (Cox), or "linear" (linear regression)
resolution	Resolution parameter for Leiden clustering (default: 0.6)
alpha	Alpha parameter for SGL balancing L1 and L2 penalties (default: 0.5)
nfold	Number of folds for cross-validation (default: 5)
dge_analysis	List controlling differential expression analysis: • run: Whether to run DEG analysis (default: FALSE) • logFC_threshold: Log fold change threshold (default: 1) • pval_threshold: P-value threshold (default: 0.05)
...	Additional arguments passed to preprocessing functions, e.g.: • verbose: Whether to print progress messages (default: TRUE) • seed: Random seed for reproducibility (default: 123L) • assay: Assay to use for clustering (default: "RNA")

Value

A list containing:

- **scRNA_data**: Seurat object with LP-SGL results integrated
- **sgl_fit**: Fitted SGL model object
- **cvfit**: Cross-validation results
- **dge_res**: Differential expression results if requested (NULL otherwise)

References

Li J, Zhang H, Mu B, Zuo H, Zhou K. Identifying phenotype-associated subpopulations through LP_SGL. Briefings in Bioinformatics. 2023 Nov 22;25(1):bbad424.

See Also

Other screen_method: DoDEGAS(), DoPIPET(), DoPhenoMapR(), DoSCIPAC(), DoSIDISH(), DoScissor(), DoTiRank(), DoscAB(), DoscPAS(), DoscPP()

Other LP_SGL: ValidateLPSGLParams()

Examples

```
## Not run:
# Example using simulated data
set.seed(123)

# Create simulated data
bulk_data <- matrix(rnorm(1000*50), nrow=1000, ncol=50)
sc_data <- matrix(rnorm(1000*500), nrow=1000, ncol=500)
phenotype <- rep(c(0, 1), each=25)

# Run LP-SGL analysis
results <- DoLP_SGL(
matched_bulk = bulk_data,
sc_data = sc_data,
phenotype = phenotype,
family = "logit",
resolution = 0.6,
dge_analysis = list(run = TRUE, logFC_threshold = 1, pval_threshold = 0.05)
)

# Access results
lpsgl_seurat <- results$scRNA_data
sgl_model <- results$sgl_fit
deg_results <- results$dge_res

## End(Not run)
```

DoPhenoMapR *PhenoMapR: Bulk-to-Single-Cell Phenotype Mapping*

Description

Derives a gene-level reference z-score from matched bulk expression and a phenotype (binary, continuous, or survival), then scores single cells and assigns tail phenotype groups via PhenoMapR.

Usage

```
DoPhenoMapR(
  matched_bulk,
  sc_data,
  phenotype,
  label_type = "PhenoMapR",
  phenotype_class = c("binary", "survival", "continuous"),
  phenotype_column = NULL,
  survival_time = "time",
  survival_event = "status",
  normalize = TRUE,
  platform = c("auto", "rnaseq", "microarray"),
  probe_annotation = NULL,
  hugo_species = c("human", "mouse"),
  binary_positive_reference = c("second", "first"),
  assay = "RNA",
  layer = "data",
  z_score_cutoff = 2L,
  reference_sign = 1L,
  score_mode = c("weighted_sum", "activity_adjusted"),
  vars_to_regress = c("S.Score", "G2M.Score", "nCount_RNA"),
  cancer_type = NULL,
  pseudobulk = FALSE,
  group_by = NULL,
  sample_metadata = NULL,
  percentile = 0.05,
  permutation_n = NULL,
  permutation_seed = NULL,
  ...,
  seed = SigBridgeRUtils::getFuncOption("seed"),
  verbose = SigBridgeRUtils::getFuncOption("verbose")
)
```

Arguments

matched_bulk	Matrix or data frame of bulk expression (genes x samples). Column names must match phenotype names/row names.
sc_data	A Seurat object containing the cells to be scored.

phenotype — Phenotype data. A named vector for binary/continuous traits, or a data frame with time-to-event and event columns for survival traits; names/row names must match `colnames(matched_bulk)`.

label_type — Character. `meta.data` column receiving the phenotype group. Default: `"PhenoMapR"`.

phenotype_class — One of `"binary"`, `"survival"`, or `"continuous"`.

phenotype_column — Character. Column name in `phenotype` for the outcome. For `phenotype_type = "survival"` this is ignored; use `survival_time` and `survival_event` instead.

survival_time — Character. Column name in `phenotype` for time-to-event (e.g. overall survival time). Required when `phenotype_type = "survival"`.

survival_event — Character. Column name in `phenotype` for event indicator (0/1 or FALSE/TRUE). Required when `phenotype_type = "survival"`.

normalize — Logical. If `TRUE`, run normalization when expression looks like counts (default `TRUE`). Set `FALSE` to skip.

platform — One of `"auto"`, `"rnaseq"`, or `"microarray"`. Microarray inputs follow the PRECOG preprocessing strategy: optional probe-to-gene mapping, log2 transform when needed, removal of zero-variance genes and samples/genes with heavy missingness, per-study quantile normalization, and per-gene unit variance scaling across samples. RNA-seq inputs use log2(CPM+1) when the matrix looks like raw counts.

probe_annotation — Optional data.frame with probe IDs and gene symbols (required for probe-level microarray matrices unless row names are already gene symbols). See collapse_probes_to_genes.

hugo_species — Character. Species for HUGO symbol cleaning: `"human"` or `"mouse"` (default `"human"`).

binary_positive_reference — For `phenotype_type "binary"` only: which level of the binary factor should correspond to the **positive** outcome in logistic regression (`y = 1`), so that genes with **positive** z-scores are those whose **higher** expression is associated with that level. Use `"first"` when the first level of `factor(..., levels = c(...))` is your phenotype of interest (e.g. `mutated` vs `wt`); use `"second"` for the legacy convention (second level coded as `y = 1`). Default `"second"` preserves behaviour from previous versions when levels were implicit (e.g. alphabetical).

assay — Assay name for Seurat/SCE objects (default: "RNA" for sc, "Spatial" for spatial)

layer — Seurat matrix to score (`"data"` normalized, `"counts"` raw, or `"scale.data"`; default `"data"`). Seurat v5 terminology. PhenoMapR maps this to `GetAssayData(layer=...)` or `GetAssayData(slot=...)` based on the installed SeuratObject version.

z_score_cutoff — Absolute z-score threshold for filtering genes (default: 2). Built-in PRECOG/TCGA matrices shipped in the package are already sparse at this cutoff (entries with $|z| \leq 2$ stored as `NA`; see precog, tcga), so lowering the cutoff below 2 does not recover discarded associations unless a denser custom reference is supplied.

reference_sign — Multiply all reference gene z-scores by this value before the weighted sum. Use `1` (default) or `-1` to flip the sign of resulting PhenoMap scores without changing the reference matrix (e.g. if you want positive scores to mean the opposite

association). For custom references from derive_reference_from_bulk, prefer setting binary_positive_reference so the reference itself matches your factor levels (first level = positive association when "first").

score_mode — Scoring mode: "weighted_sum" (default) uses the standard dot product of expression and reference z-scores; "activity_adjusted" regresses technical covariates per gene (via Seurat ScaleData) before computing the weighted sum on the regressed layer. Requires Seurat for matrix inputs.

vars_to_regress — Metadata columns to regress when score_mode = "activity_adjusted". Default: c("S.Score", "G2M.Score", "nCount_RNA"). For matrix input, pass covariates via sample_metadata (see below). nCount_RNA is computed from column sums when absent.

cancer_type — Cancer type label. Required if using built-in reference datasets. Should match column names in reference data. For ICI PRECOG, use format "ABBREV" or "ABBREV_Metastatic" (e.g., "MELANOMA", "MELANOMA_Metastatic")

pseudobulk — Logical. If TRUE, aggregate expression before scoring (default: FALSE)

group_by — Column name for pseudobulk grouping. Required if pseudobulk = TRUE. For Seurat/SCE objects, should be a metadata column.

sample_metadata — Optional data.frame of per-cell/sample covariates for score_mode = "activity_adjusted" when expression is a matrix. Must include a cell identifier column (default "cell_id").

percentile — Fraction of cells in each tail (default 0.05 for top and bottom 5%).

permutation_n — If a positive integer, compute gene-shuffle permutation null scores and add empirical_p_* columns alongside raw score columns.

permutation_seed — Random seed for permutation null (default 42).

... — Additional arguments reserved for method compatibility.

seed — Integer. Random seed. Default: getFuncOption("seed").

verbose — Logical. Whether to print progress messages. Default: getFuncOption("verbose").

Details

define_phenotype_groups() labels the upper tail of the score distribution "Most Adverse" and the lower tail "Most Favorable". The score increases towards the phenotype-valued-as-1 end when the reference is derived with the default orientation (for a numeric 0/1 phenotype, binary_positive_reference = "second" regresses on the "1" level, and for survival traits the Cox z-score is positive for hazard-increasing genes). Therefore, when phenotype 1 denotes the unfavourable direction, the "Most Adverse" tail is the one to be labelled "Positive"; the group labels are mapped to the package-wide vocabulary as "Most Adverse" -> "Positive", "Most Favorable" -> "Negative" and "Other" -> "Neutral". Setting reference_sign = -1 mirrors the score axis, in which case the two tails swap. If a phenotype is encoded so that the adverse class is *not* the reference-positive level (e.g. binary_positive_reference = "first" with numeric 0/1 labels), the score direction is reversed as well; choose the arguments so that higher scores track the adverse phenotype.

Value

A list containing:

- scRNA_data: Seurat object with label_type ("Positive", "Negative", "Neutral", or NA) and a PhenoMapR_* score column in meta.data.
- phenomap_scores: Per-cell scores from PhenoMap.
- reference: Gene-level reference z-scores.

References

PhenoMapR: mapping phenotypic information from bulk to single-cell data. https://github.com/brooksbenard/PhenoMapR

See Also

Other screen_method: DoDEGAS(), DoLP_SGL(), DoPIPET(), DoSCIPAC(), DoSIDISH(), DoScissor(), DoTiRank(), DoscAB(), DoscPAS(), DoscPP()

Other PhenoMapR: ValidatePhenoMapRParams()

Examples

```
## Not run:
res <- DoPhenoMapR(
  matched_bulk = bulk_matrix,
  sc_data = seurat_obj,
  phenotype = a_named_vector,
  label_type = "PhenoMapR",
  phenotype_class = "binary"
)

## End(Not run)
```

DoPIPET *Perform PIPET Screening Analysis*

Description

Predicts cell subpopulations in single-cell data by matching expression profiles to predefined marker gene templates using various distance/similarity metrics. This function implements a template-based classification approach with permutation testing for significance assessment.

Usage

```
DoPIPET(
  matched_bulk,
  sc_data,
  phenotype,
  phenotype_class = c("binary", "continuous", "survival"),
  only_pos_marker = FALSE,
  group = NULL,
  discretize_method = c("kmeans", "median", "custom"),
  cutoff = NULL,
  label_type = "PIPET",
  marker_finder = c("limma", "DESeq2"),
  log2FC = 1L,
  p_adjust = 0.05,
  show_log2FC = TRUE,
  freq_counts = NULL,
  normalize = TRUE,
  scale = TRUE,
  nPerm = 1000L,
 distance = c("cosine", "pearson", "spearman", "kendall", "euclidean", "maximum"),
  ...
)
```

Arguments

matched_bulk	Normalized bulk expression matrix (features x samples). Column names must match phenotype identifiers.
sc_data	Seurat object containing single-cell RNA-seq data.
phenotype	Clinical outcome data. Can be: • Vector: named with sample IDs • Data frame: with row names matching bulk columns
phenotype_class	Analysis mode: • "binary": Case-control design (e.g., responder/non-responder) • "continuous": Continuous outcome (e.g., age, size) • "survival": Patient survival
only_pos_marker	A logical value. If TRUE, only upregulated marker genes (those with positive log2 fold change) will be retained. If FALSE (default), both upregulated and downregulated markers are kept.
group	A character, name of one metadata column to group cells by (for example, orig.ident). The default value is NULL. In this case, screening will be performed on each group separately.
discretize_method	c("median", "kmeans", "custom"). Discretization strategy for continuous phenotypes. Note: "median" is mapped internally to "quantile" (2-group quantile split). Default: "kmeans".

cutoff	Numeric vector of length n_group - 1. Required only when discretize_method = "custom". Defines interior breakpoints on the *normalized, log2-transformed scale* (i.e., after scale(log2(x + 1))). Must be sorted in ascending order.
label_type	Character specifying phenotype label type (e.g., "SBS1", "time"), stored in scRNA_data@misc
marker_finder	A character, the marker finder method. The default value is "limma".
log2FC	In the DESeq differential expression analysis results, the cutoff value of log2FC. The default value is 1L.
p_adjust	In the DESeq differential expression analysis results, the cutoff value of adjust P. The default value is 0.05.
show_log2FC	Select whether to show log2 fold changes. The default value is TRUE.
freq_counts	An integer, keep genes expressed in more than a certain number of cells. The default value is NULL, which means no filtering.
normalize	Select whether to perform normalization of count data. The default value is TRUE.
scale	Select whether to scale and center features in the dataset. The default value is TRUE.
nPerm	An integer, number of permutations to do. The default value is 1000L.
distance	A character, the distance algorithm must be included in "cosine", "pearson", "spearman", "kendall","euclidean","maximum". default value is "cosine".
...	Additional arguments to be passed to PIPET.optimized. • seed: Random seed for reproducibility • verbose: Whether to show progress messages • parallel: Whether to use parallel processing, default is FALSE. future::plan() must be set before calling this function. • assay: The assay to use, default is "RNA"

See Also

Other screen_method: DoDEGAS(), DoLP_SGL(), DoPhenoMapR(), DoSCIPAC(), DoSIDISH(), DoScissor(), DoTiRank(), DoscAB(), DoscPAS(), DoscPP()

Other PIPET: ValidatePIPETParams()

DoscAB *Perform scAB Screening Analysis*

Description

Implements the scAB algorithm to identify phenotype-associated cell subpopulations in single-cell RNA-seq data by integrating matched bulk expression and phenotype information. Uses non-negative matrix factorization (NMF) with dual regularization for phenotype association and cell-cell similarity.

Usage

```
DoscAB(
  matched_bulk,
  sc_data,
  phenotype,
  label_type = "scAB",
  phenotype_class = c("binary", "survival"),
  alpha = c(0.005, NULL),
  alpha_2 = c(0.005, NULL),
  k_max = 20L,
  cross_k = 5L,
  repeat_times = 10L,
  maxiter = 2000L,
  tred = 2L,
  ...
)
```

Arguments

matched_bulk	Matrix or data frame of preprocessed bulk RNA-seq expression data (genes x samples). Column names must match names/IDs in phenotype.
sc_data	A Seurat object containing scRNA-seq data to be screened.
phenotype	Phenotype data, either: • Named vector, where names match columns of matched_bulk • Patient survival data frame, where row names match columns of matched_bulk, and column names are "time" and "status"
label_type	Character specifying phenotype label type, e.g. "SBS1" or "time".
phenotype_class	Type of phenotypic outcome. Must be consistent with input data: • "binary": Binary traits, e.g. case/control • "continuous": Continuous measurements • "survival": Survival information
alpha	Coefficient of phenotype regularization (default=0.005).
alpha_2	Coefficent of cell-cell similarity regularization (default=0.005).
k_max	The maximum rank value to consider in the search. Must be at least 2. Defaults to 20.
cross_k	Number of folds for k-fold cross-validation. Defaults to 5.
repeat_times	The number of repeated NMF runs for each candidate rank to account for random initialization variability. Defaults to 10.
maxiter	Maximum number of iterations for NMF (default=2000).
tred	Z-score threshold in finding subsets (default=2).
...	Additional arguments. Currently supports: • verbose: Logical indicating whether to print progress messages. Defaults to TRUE.

- seed: For reproducibility, default is 123L
- parallel: Logical indicating whether to use parallel processing. Defaults to FALSE.
- Other arguments are passed to scAB::create_scAB.v5()
- assay: Character specifying the assay to use. Defaults to "RNA".
- load_cache: Cache directory path for loading precomputed scAB objects. Supports root-level, cache-level, or parent-level paths. See CacheSetHere().
- save_cache: Cache directory path for saving scAB intermediate results. Supports root-level or parent-level paths. See CacheSetHere().

Value

A list containing:

- **scRNA_data**: Filtered Seurat object with selected cells
- **scAB_result**: scAB screening result

LICENSE

Licensed under the GNU General Public License version 3 (GPL-3.0). A copy of the license is available at https://www.gnu.org/licenses/gpl-3.0.en.html.

References

Zhang Q, Jin S, Zou X. scAB detects multiresolution cell states with clinical significance by integrating single-cell genomics and bulk sequencing data. Nucleic Acids Research. 2022 Nov 28;50(21):12112-30.

See Also

Other screen_method: DoDEGAS(), DoLP_SGL(), DoPIPET(), DoPhenoMapR(), DoSCIPAC(), DoSIDISH(), DoScissor(), DoTiRank(), DoscPAS(), DoscPP()

Other scAB: ValidatescABParams()

Examples

```
## Not run:
# Binary phenotype example
result <- DoscAB(
  matched_bulk = bulk_matrix,
  sc_data = seurat_obj,
  phenotype = clinical_df,
  label_type = "disease_status",
  phenotype_class = "binary",
  alpha = 0.005,
  alpha_2 = 0.005,
  maxiter = 2000,
  tred = 2
)
```

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

DoSCIPAC *Screen Single-Cell Data Using SCIPAC Algorithm*

Description

Performs single-cell phenotype association analysis using the SCIPAC method. Integrates bulk RNA-seq phenotype information with single-cell transcriptomic data to identify cell populations associated with clinical outcomes.

Usage

```
DoSCIPAC(
  matched_bulk,
  sc_data,
  phenotype,
  label_type = "SCIPAC",
  phenotype_class = c("binary", "survival", "continuous"),
  hvg = 1000L,
  do_pca_sc = FALSE,
  n_pc = 60L,
  sc_batch_col = NULL,
  resolution = 2L,
  ela_net_alpha = 0.4,
  bt_size = 50L,
  ncore = 7L,
  ci_alpha = 0.05,
  nfold = 10L,
  ...
)
```

Arguments

matched_bulk	Matrix or data frame of preprocessed bulk RNA-seq expression data (genes x samples). Column names must match names/IDs in phenotype.
sc_data	A Seurat object containing scRNA-seq data to be screened.
phenotype	Phenotype data, either: • Named vector (names match matched_bulk columns) • Data frame with row names matching matched_bulk columns, containing "time" and "status" columns for survival analysis
label_type	Character. Context info specifying the screening method identifier. Default: "SCIPAC".
phenotype_class	Type of phenotypic outcome:

- "binary": Binary traits (e.g., case/control, responder/non-responder)
- "continuous": Continuous measurements (e.g., age, biomarker levels)
- "survival": Survival information (time-to-event data)

Partial matching is supported.

hvg	Integer. Number of highly variable genes to use for preprocessing. Default: 1000L.
do_pca_sc	Logical. if TRUE, first do PCA on sc.dat and use the rotation matrix on bulk.dat; if False, first do PCA on bulk.data and use the rotation matrix on sc.dat. The default is FALSE.
n_pc	Integer. Number of principal components to use. Default: 60L.
sc_batch_col	Character or vector. Batch variable for single-cell data. If character, should be a column name in sc_data @metadata. If vector, should be a batch assignment vector matching cell order. Default: NULL (no batch correction).
resolution	Numeric. Clustering resolution for cell type identification. Higher values produce more clusters. Default: 2L.
ela_net_alpha	Numeric. Elastic net mixing parameter (0 = ridge, 1 = lasso). Default: 0.4.
bt_size	Integer. Bootstrap sample size for stability assessment. Default: 50L.
ncore	Integer. Number of CPU cores for parallel computation. Default: 7L.
ci_alpha	Numeric. Significance level for confidence intervals. Default: 0.05.
nfold	Integer. Number of folds for cross-validation for regression models. Default: 10L.
...	Additional arguments (support assay(character), verbose(logical) & seed(integer)).

Value

A named list containing:

- scRNA_data: Modified Seurat object with SCIPAC screening results added as metadata columns. Includes association statistics, p-values, and confidence intervals for each cell cluster.
- pca_res: PCA rotation results from SCIPAC::sc.bulk.pca()
- cluster_res: Clustering results from SCIPAC::seurat.ct()

Workflow

1. **Gene overlap**: Identifies common genes between scRNA-seq and bulk data
2. **Preprocessing**: Selects HVGs and preprocesses both datasets
3. **PCA rotation**: Aligns scRNA-seq and bulk data in shared PCA space
4. **Clustering**: Identifies cell populations using Seurat clustering
5. **Association testing**: Tests cluster-phenotype associations using elastic net regression with bootstrap validation
6. **Result integration**: Adds screening results to Seurat object metadata

Requirements

- R package: SCIPAC (install from Exceret/SCIPAC)
- Gene symbols must be consistent between scRNA-seq and bulk datasets
- Sufficient overlapping genes (typically >100) for reliable rotation

Notes

- For survival analysis, phenotype must be a data frame with "time" and "status" columns
- Binary phenotypes are automatically converted to factors
- Continuous phenotypes use Gaussian family in elastic net regression
- Screening parameters are stored in scRNA_data@misc$SCIPAC_params for reproducibility

See Also

Other screen_method: DoDEGAS(), DoLP_SGL(), DoPIPET(), DoPhenoMapR(), DoSIDISH(), DoScissor(), DoTiRank(), DoscAB(), DoscPAS(), DoscPP()

Examples

```
## Not run:
# Example 1: Binary phenotype (case/control)
library(SCIPAC)

# Prepare data
bulk_expr <- matrix(rpois(1000, 10), nrow = 200, ncol = 5)
colnames(bulk_expr) <- paste0("Sample", 1:5)
rownames(bulk_expr) <- paste0("Gene", 1:200)

seurat_obj <- CreateSeuratObject(counts = matrix(rpois(2000, 5), nrow = 200, ncol = 10))

phenotype_binary <- c(Sample1 = 1, Sample2 = 0, Sample3 = 1, Sample4 = 0, Sample5 = 1)

# Run SCIPAC screening
result <- DoSCIPAC(
  matched_bulk = bulk_expr,
  sc_data = seurat_obj,
  phenotype = phenotype_binary,
  phenotype_class = "binary",
  hvg = 100,
  resolution = 1,
  verbose = TRUE
)

# Access results
screened_seurat <- result$scRNA_data
head(screened_seurat[[]])

# Example 2: Survival analysis
survival_data <- data.frame(
  time = c(12.5, 24.3, 18.7, 36.2, 30.1),
```

```
  status = c(1, 0, 1, 1, 0),
  row.names = paste0("Sample", 1:5)
)

result <- DoSCIPAC(
  matched_bulk = bulk_expr,
  sc_data = seurat_obj,
  phenotype = survival_data,
  phenotype_class = "survival",
  ela_net_alpha = 0.5,
  bt_size = 100,
  ncore = 4
)

# Example 3: Continuous phenotype with custom parameters
continuous_pheno <- c(Sample1 = 25.3, Sample2 = 45.7, Sample3 = 38.2,
                      Sample4 = 52.1, Sample5 = 31.8)

result <- DoSCIPAC(
  matched_bulk = bulk_expr,
  sc_data = seurat_obj,
  phenotype = continuous_pheno,
  phenotype_class = "continuous",
  hvg = 500,
  n_pc = 30,
  resolution = 2,
  ela_net_alpha = 0.3,
  ci_alpha = 0.01,
  seed = 42
)

# View screening parameters used
result$scRNA_data@misc$SCIPAC_params

# Example 4: With batch correction
seurat_obj$batch_id <- sample(c("Batch1", "Batch2"), ncol(seurat_obj), replace = TRUE)

result <- DoSCIPAC(
  matched_bulk = bulk_expr,
  sc_data = seurat_obj,
  phenotype = phenotype_binary,
  phenotype_class = "binary",
  sc_batch_col = "batch_id",
  ncore = 8
)

## End(Not run)
```

DoScissor *Perform Scissor Screening Analysis*

Description

Identifies phenotype-associated cell subpopulations in single-cell data using regularized regression on matched bulk expression profiles. Scissor integrates bulk and single-cell RNA-seq data to identify cells that are significantly associated with phenotypic outcomes.

Usage

```
DoScissor(
  matched_bulk,
  sc_data,
  phenotype,
  label_type = "scissor",
  phenotype_class = c("binary", "continuous", "survival"),
  alpha = c(0.05, NULL),
  cutoff = 0.2,
  family = lifecycle::deprecated(),
  reliability_test = list(run = FALSE, n = 10L, nfold = 10L),
 cell_evaluation = list(run = FALSE, benchmark_data = "path_to_file.RData", FDR_cutoff =
    0.05, bootstrap_n = 100L),
  ...
)
```

Arguments

matched_bulk	Matrix or data frame of preprocessed bulk RNA-seq expression data (genes x samples). Column names must match names/IDs in phenotype.
sc_data	A Seurat object containing scRNA-seq data to be screened.
phenotype	Phenotype data, either: • Named vector, where names match columns of matched_bulk • Patient survival data frame, where row names match columns of matched_bulk, and column names are "time" and "status"
label_type	Character specifying phenotype label type, e.g. "SBS1" or "time".
phenotype_class	Type of phenotypic outcome. Must be consistent with input data: • "binary": Binary traits, e.g. case/control • "continuous": Continuous measurements • "survival": Survival information
alpha	Parameter used to balance the effect of the l1 norm and the network-based penalties. It can be a number or a searching vector. If alpha = NULL, a default searching vector is used. The range of alpha is between 0 and 1. A larger alpha lays more emphasis on the l1 norm.
cutoff	(default: 0.2). When alpha=NULL, the cutoff is used to determine the optimal alpha. Higher values increase specificity.
family	**[Deprecated]** Model family for outcome type: • "gaussian": Continuous outcomes

- "binomial": Binary outcomes (default)
- "cox": Survival outcomes

`reliability_test`
: List controlling reliability testing:
  - run: Whether to perform reliability test (default: FALSE)
  - n: Permutation times (default: `10L`)
  - nfold: Cross-validation folds (default: `10L`)

`cell_evaluation`
: List controlling cell evaluation:
  - run: Whether to perform cell evaluation (default: FALSE)
  - benchmark_data: Path to benchmark data (RData file)
  - FDR_cutoff: FDR threshold for evaluation (default: `0.05`)
  - bootstrap_n: Bootstrap iterations (default: `100L`)

`...`
: Additional arguments. Currently supports:
  - `verbose`: Logical indicating whether to print progress messages. Defaults to `TRUE`.
  - `seed`: For reproducibility, default is `123L`
  - `assay`: Assay to use for single-cell data. Defaults to `"RNA"`
  - `load_cache`: Cache directory path for loading precomputed Scissor inputs. Supports root-level, cache-level, or parent-level paths. See `CacheSetHere()`.
  - `save_cache`: Cache directory path for saving Scissor inputs. Supports root-level or parent-level paths. See `CacheSetHere()`.
  - `path2load_scissor_cache` / `path2save_scissor_inputs`: Deprecated names, kept for backward compatibility.

Value

A list containing:

- **scRNA_data**: A Seurat object with screened cells containing metadata:
  - **scissor**: "Positive"/"Negative"/"Neutral" classification
  - **label_type**: Outcome label used
- **scissor_result**: Raw Scissor results
- **reliability_result**: If reliability_test=TRUE, contains:
  - **statistic**: A value between 0 and 1
  - **p**: p-value of the test statistic
  - **AUC_test_real**: 10 values of AUC for real data
  - **AUC_test_back**: A list of AUC for background data
- **cell_evaluation**: If cell_evaluation=TRUE, contains:
  - **evaluation_res**: A data.frame with some supporting information for each Scissor selected cell

LICENSE

Licensed under the GNU General Public License version 3 (GPL-3.0). A copy of the license is available at https://www.gnu.org/licenses/gpl-3.0.en.html.

References

Sun D, Guan X, Moran AE, Wu LY, Qian DZ, Schedin P, et al. Identifying phenotype-associated subpopulations by integrating bulk and single-cell sequencing data. Nat Biotechnol. 2022 Apr;40(4):527-38.

See Also

Other screen_method: DoDEGAS(), DoLP_SGL(), DoPIPET(), DoPhenoMapR(), DoSCIPAC(), DoSIDISH(), DoTiRank(), DoscAB(), DoscPAS(), DoscPP()

Other scissor: DoScissorCellEval(), DoScissorRelTest(), ValidateScissorParams()

Examples

```
## Not run:
# Binary outcome example
res <- DoScissor(
  matched_bulk = bulk_matrix,
  sc_data = seurat_obj,
  phenotype = a_named_vector,
  family = "binomial"
)

## End(Not run)
```

DoscPAS *Perform scPAS Screening Analysis*

Description

This function performs scPAS screening analysis by integrating bulk and single-cell RNA-seq data. It includes data filtering steps and wraps the core scPAS::scPAS function.

Usage

```
DoscPAS(
  matched_bulk,
  sc_data,
  phenotype,
  label_type = "scPAS",
  phenotype_class = c("binary", "continuous", "survival"),
  assay = "RNA",
  imputation = FALSE,
```

```
  imputation_method = c("KNN", "ALRA"),
  nfeature = 3000L,
  alpha = c(0.01, NULL),
  cutoff = 0.2,
  network_class = c("SC", "bulk"),
  family = lifecycle::deprecated(),
  permutation_times = 2000L,
  FDR_threshold = 0.05,
  independent = TRUE,
  ...
)
```

Arguments

matched_bulk	Matrix or data frame of preprocessed bulk RNA-seq expression data (genes x samples). Column names must match names/IDs in phenotype.
sc_data	A Seurat object containing scRNA-seq data to be screened.
phenotype	Phenotype data, either: • Named vector, where names match columns of matched_bulk • Patient survival data frame, where row names match columns of matched_bulk, and column names are "time" and "status"
label_type	Character specifying phenotype label type, e.g. "SBS1" or "time".
phenotype_class	Type of phenotypic outcome. Must be consistent with input data: • "binary": Binary traits, e.g. case/control • "continuous": Continuous measurements • "survival": Survival information
assay	Assay to use from sc_data (default: 'RNA')
imputation	Logical, whether to perform imputation (default: FALSE)
imputation_method	Character. Name of alternative method for imputation. (options: "KNN", "ALRA")
nfeature	Number of features to select (default: 3000, indicating that the top 3000 highly variable genes are selected for model training
alpha	Numeric. Significance threshold. Parameter used to balance the effect of the l1 norm and the network-based penalties. It can be a number or a searching vector. If alpha = NULL, a default searching vector is used. The range of alpha is in `[0,1]`. A larger alpha lays more emphasis on the l1 norm. (default: 0.01)
cutoff	Numeric. Cutoff value for selecting the optimal alpha value when alpha = NULL. (default: 0.2)
network_class	Network class to use (default: 'SC', indicating gene-gene similarity networks derived from single-cell data. The other one is 'bulk'.)
family	**[Deprecated]** Model family for analysis (options: "cox", "gaussian", "binomial")

`permutation_times`
: Number of permutations to perform (default: 2000)

`FDR_threshold`
: Numeric. FDR value threshold for identifying phenotype-associated cells (default: 0.05)

`independent`
: Logical. The background distribution of risk scores is constructed independently of each cell. (default: TRUE)

`...`
: Additional arguments. Currently supports:

- `verbose`: Logical indicating whether to print progress messages. Defaults to `TRUE`.
- `seed`: For reproducibility, default is `123L`

Value

A Seurat object from scPAS analysis

LICENSE

Licensed under the GNU General Public License version 3 (GPL-3.0). A copy of the license is available at https://www.gnu.org/licenses/gpl-3.0.en.html.

References

Xie A, Wang H, Zhao J, Wang Z, Xu J, Xu Y. scPAS: single-cell phenotype-associated subpopulation identifier. Briefings in Bioinformatics. 2024 Nov 22;26(1):bbae655.

See Also

Other screen_method: `DoDEGAS()`, `DoLP_SGL()`, `DoPIPET()`, `DoPhenoMapR()`, `DoSCIPAC()`, `DoSIDISH()`, `DoScissor()`, `DoTiRank()`, `DoscAB()`, `DoscPP()`

Other scPAS: `ValidatescPASParams()`

DoscPP *Perform scPP screening analysis*

Description

This function performs scPP screening on single-cell data using matched bulk data and phenotype information. It supports binary, continuous, and survival phenotype types.

Usage

```
DoscPP(
  matched_bulk,
  sc_data,
  phenotype,
  label_type = "scPP",
  phenotype_class = c("binary", "continuous", "survival"),
  ref_group = 0L,
  Log2FC_cutoff = 0.585,
  estimate_cutoff = 0.2,
  probs = c(0.2, NULL),
  ...
)
```

Arguments

matched_bulk	Matrix or data frame of preprocessed bulk RNA-seq expression data (genes x samples). Column names must match names/IDs in phenotype.
sc_data	A Seurat object containing scRNA-seq data to be screened.
phenotype	Phenotype data, either: • Named vector, where names match columns of matched_bulk • Patient survival data frame, where row names match columns of matched_bulk, and column names are "time" and "status"
label_type	Character specifying phenotype label type, e.g. "SBS1" or "time".
phenotype_class	Type of phenotypic outcome. Must be consistent with input data: • "binary": Binary traits, e.g. case/control • "continuous": Continuous measurements • "survival": Survival information
ref_group	Reference group or baseline for **binary** comparisons, e.g. "Normal" for Tumor/Normal studies and 0 for 0/1 case-control studies. (default: 0)
Log2FC_cutoff	Minimum log2 fold-change for binary markers (default: 0.585)
estimate_cutoff	Effect size threshold for **continuous** traits (default: 0.2)
probs	A numeric value indicating the quantile cutoff for cell classification. This parameter can also be a numeric vector, in which case an optimal threshold will be selected based on the AUC and enrichment score.(default: `0.2`)
...	Additional arguments. Currently supports: • `verbose`: Logical indicating whether to print progress messages. Defaults to `TRUE`. • `seed`: For reproducibility, default is `123L` • `assay`: The assay to use for single-cell data. Defaults to `RNA`.

Value

A list containing:

- **scRNA_data**: Seurat object with added metadata:
  - **ScPP**: "Positive"/"Negative"/"Neutral" classification
- **gene_list**: List of genes used for screening

Algorithm Steps

1. Data Validation: Checks sample alignment between bulk and phenotype data
2. Marker Selection: Identifies phenotype-associated genes from bulk data
3. Single-cell Screening: Projects bulk markers onto single-cell data
4. Cell Classification: Categorizes cells based on phenotype association

Reference

WangX-Lab/ScPP [Internet]. [cited 2025 Aug 31]. Available from: https://github.com/WangX-Lab/ScPP

See Also

Other screen_method: `DoDEGAS()`, `DoLP_SGL()`, `DoPIPET()`, `DoPhenoMapR()`, `DoSCIPAC()`, `DoSIDISH()`, `DoScissor()`, `DoTiRank()`, `DoscAB()`, `DoscPAS()`

Other scPP: `ValidatescPPParams()`

Examples

```
## Not run:
# Binary phenotype analysis
res <- DoscPP(
  matched_bulk = bulk_data,
  sc_data = seurat_obj,
  phenotype = ms_data,
  label_type = "SBS1",
  phenotype_class = "Binary"
)

# Survival analysis
surv_res <- DoscPP(
  sc_data = seurat_obj,
  matched_bulk = bulk_data,
  phenotype = surv_df,
  label_type = "OS_status",
  phenotype_class = "Survival"
)

## End(Not run)
```

DoSIDISH *Perform SIDISH Screening Analysis*

Description

Perform SIDISH Screening Analysis

Usage

```
DoSIDISH(
  matched_bulk,
  sc_data,
  phenotype,
  label_type = "SIDISH",
  phenotype_class = "survival",
  sidish_params = list(),
  ...
)
```

Arguments

matched_bulk	Matrix or data frame of preprocessed bulk RNA-seq expression data (genes x samples). Column names must match names/IDs in phenotype.
sc_data	A Seurat object containing scRNA-seq data to be screened.
phenotype	Phenotype data, either: • Patient survival Data frame with row names matching matched_bulk columns, colnames named "time" and "status"
label_type	Character specifying phenotype label type
phenotype_class	Type of phenotypic outcome (must be consistent with input data): • "survival": Survival infomation
sidish_params	List of SIDISH algorithm parameters including: **Preprocessing parameters:** • patient_id: column name for patient identifier in metadata (default: "Sample") • celltype_name: column name for cell type annotation in metadata (default: "celltype_major") • processed: whether input data is already preprocessed (default: TRUE) • n_genes_by_counts: minimum number of genes expressed per cell filter threshold (default: 5000) • pct_counts_mt: maximum percentage of mitochondrial genes filter threshold (default: 10) • batch_correction: whether to perform batch correction (default: FALSE) • survival_: column name for survival time in phenotype data (default: "time")

- status: column name for event status in phenotype data (default: "status")

Execution environment:

- device: computation device, "cuda" for GPU acceleration or "cpu" for CPU-only (default: "cuda")
- use_spatial_graph: whether to use spatial graph information (default: FALSE)
- k_neighbors: number of neighbors for graph construction (default: NULL, auto-detected)

Phase 1: VAE training parameters:

- phase1_epochs: total epochs for VAE training (default: 225)
- phase1_i_epochs: interval epochs for VAE intermediate evaluation (default: 20)
- phase1_latent_size: dimensionality of latent space (default: 32)
- phase1_layer_dims: hidden layer dimensions as integer vector (default: c(512, 128))
- phase1_batch_size: batch size for VAE training (default: 256)
- phase1_optimizer: optimizer algorithm (default: "Adam")
- phase1_lr: learning rate for VAE encoder/decoder (default: 1e-4)
- phase1_lr_3: learning rate for additional VAE component (default: 1e-4)
- phase1_dropout: dropout rate for VAE layers (default: 0)
- phase1_type: VAE layer type, "Dense" or "Normal" (default: "Dense")

Phase 2: Deep Cox training parameters:

- phase2_epochs: total epochs for Cox model training (default: 500)
- phase2_hidden: number of hidden units in Cox model (default: 128)
- phase2_lr: learning rate for Cox model (default: 1e-4)
- phase2_dropout: dropout rate for Cox model (default: 0)
- phase2_test_size: proportion of data held out for testing (default: 0.2)
- phase2_batch_size_bulk: batch size for bulk data in Cox training (default: 256)

Training & risk definition parameters:

- train_iterations: number of risk score iteration rounds (default: 5)
- train_percentile: percentile threshold for high-risk cell selection (default: 0.95)
- train_steepness: steepness parameter for risk score transformation (default: 30)
- train_path: directory path for saving intermediate results (default: "./SIDISH_res/")
- train_num_workers: number of data loading workers (default: 0)
- train_distribution_fit: distribution fitting method, "fitted" or "default" (default: "fitted")

... Additional arguments passed to the function. Common parameters include:

- verbose: Logical. Whether to print verbose output (default: TRUE).
- seed: Integer. Random seed for reproducibility (default: 123L).
- assay: Character. Assay to use for screening (default: "RNA").

Value

A named list containing:

- **scRNA_data**: Modified single-cell data object with integrated screening results.

See Also

Other screen_method: DoDEGAS(), DoLP_SGL(), DoPIPET(), DoPhenoMapR(), DoSCIPAC(), DoScissor(), DoTiRank(), DoscAB(), DoscPAS(), DoscPP()

Other SIDISH: SIDISHEnvSet(), ValidateSIDISHParams()

DoTiRank *Perform TiRank Screening Analysis*

Description

Integrates single-cell and bulk RNA-seq data using the TiRank deep learning framework to identify phenotype-associated cells. TiRank employs a rank-based approach with neural network models to score and classify cells based on their association with the phenotype of interest.

Usage

```
DoTiRank(
  matched_bulk,
  sc_data,
  phenotype,
  label_type = "TiRank",
  phenotype_class = c("binary", "survival", "continuous"),
 tirank_params = list(validation_proportion = 0.15, sampling_thresh = 0.5, sampling_mode
   = c("smote", "downsample", "upsample", "tomeklinks"), top_var_genes = 2000L,
  top_gene_pairs = 1000L, p_value_threshold = 0.05, max_cutoff = 0.8, min_cutoff =
  -0.8, nhead = 2L, nhid1 = 96L, nhid2 = 8L, n_output = 32L, nlayers = 3L, n_pred = 2L,
  dropout = 0.5, encoder_type = c("MLP", "Transformer", "DenseNet"), infer_mode =
   c("SC", "ST"), n_trials = 5L, do_reject = TRUE, tolerance = 0.05),
  save_path = "./TiRank_res",
  load_cache = NULL,
  ...
)
```

Arguments

matched_bulk	Matrix or data frame of preprocessed bulk RNA-seq expression data (genes x samples). Column names must match names/IDs in phenotype.
sc_data	A matrix/Matrix (genes x cells) or a Seurat object containing scRNA-seq data to be screened.
phenotype	Phenotype data, either:

- Named vector (names match `matched_bulk` columns), or
- Patient survival data frame with row names matching `matched_bulk` columns, colnames named "time" and "status"

`label_type` Character specifying phenotype label type (default: `"TiRank"`).

`phenotype_class`

Type of phenotypic outcome (must be consistent with input data):

- `"binary"`: Binary traits (e.g., case/control)
- `"continuous"`: Continuous measurements
- `"survival"`: Survival information

`tirank_params` List of TiRank algorithm parameters:

Data preprocessing:

- `validation_proportion`: Proportion of bulk data held out for validation (default: `0.15`).
- `sampling_thresh`: Threshold for resampling strategy (default: `0.5`).
- `sampling_mode`: Resampling method: `"smote"`, `"downsample"`, `"upsample"`, or `"tomeklinks"`.
- `top_var_genes`: Number of top variable genes to select (default: `2000L`).
- `top_gene_pairs`: Number of top gene pairs for ranking (default: `1000L`).
- `p_value_threshold`: P-value threshold for feature selection (default: `0.05`).
- `max_cutoff`: Upper cutoff for correlation filtering (default: `0.8`).
- `min_cutoff`: Lower cutoff for correlation filtering (default: `-0.8`).

Neural network architecture:

- `nhead`: Number of attention heads (default: `2L`).
- `nhid1`: Size of first hidden layer (default: `96L`).
- `nhid2`: Size of second hidden layer (default: `8L`).
- `n_output`: Output dimension of encoder (default: `32L`).
- `nlayers`: Number of encoder layers (default: `3L`).
- `n_pred`: Number of prediction heads (default: `2L`).
- `dropout`: Dropout rate for regularization (default: `0.5`).
- `encoder_type`: Encoder architecture: `"MLP"`, `"Transformer"`, or `"DenseNet"`.
- `infer_mode`: Inference mode: `"SC"` (single-cell) or `"ST"` (spatial transcriptomics).

Training:

- `n_trials`: Number of repeated training trials (default: `5L`).
- `do_reject`: Whether to apply rejection criteria to uncertain predictions (default: `TRUE`).
- `tolerance`: Tolerance threshold for rejection (default: `0.05`).

`save_path` (Soft-deprecated) Directory path for saving intermediate and final results (default: `"./TiRank_res"`). Acts as fallback for `load_cache` and `save_cache` when not specified via `...`. Prefer using `load_cache` / `save_cache` in `...` instead. See `CacheSetHere()`.

load_cache (Soft-deprecated) Optional path to cached data (default: NULL). Prefer using load_cache in ... instead.

... Additional arguments passed to the function. Common parameters include:

- verbose: Logical. Whether to print verbose output (default: TRUE).
- seed: Integer. Random seed for reproducibility.
- assay: Character. Name of assay to use from Seurat object (default: "RNA").
- load_cache: Cache directory path for loading cached data. Supports root-level, cache-level, or parent-level paths. See CacheSetHere().
- save_cache: Cache directory path for saving results. Supports root-level or parent-level paths. See CacheSetHere().

Details

The TiRank screening workflow consists of the following steps:

1. **Data preprocessing:** Normalizes bulk expression data and validates compatibility with phenotype information.
2. **Expression transfer:** Transfers the single-cell expression profile into the TiRank-compatible format.
3. **Validation set generation:** Splits bulk data into training and validation sets according to validation_proportion.
4. **Resampling:** Applies the specified sampling strategy to address class imbalance in bulk data.
5. **Cell-cell similarity:** Computes cell-cell distance or similarity matrix from single-cell data.
6. **Model training:** Trains a TiRank neural network (MLP, Transformer, or DenseNet encoder) using bulk expression and phenotype.
7. **Screening:** Applies the trained model to score each cell, producing rank-based predictions with rejection handling.
8. **Label assignment:** Classifies cells as "Positive", "Neutral", or "Negative" based on rank scores.

Value

A named list containing:

- **scRNA_data**: Modified single-cell data object with integrated screening results added as metadata, including TiRank_Reject, TiRank_Rank_Score, and TiRank (Positive/Neutral/Negative) columns, plus TiRank_para and TiRank_type stored in misc slot.
- **cell_cell_distance**: Computed cell-cell similarity/distance matrix.

See Also

Other TiRank: ValidateTiRankParams()

Other screen_method: DoDEGAS(), DoLP_SGL(), DoPIPET(), DoPhenoMapR(), DoSCIPAC(), DoSIDISH(), DoScissor(), DoscAB(), DoscPAS(), DoscPP()

Examples

```
## Not run:
# Binary classification example
result <- DoTiRank(
  matched_bulk = bulk_matrix,
  sc_data = seurat_obj,
  phenotype = bulk_phenotype,
  phenotype_class = "binary",
  save_path = "./TiRank_res"
)

# Survival analysis example
result <- DoTiRank(
  matched_bulk = bulk_matrix,
  sc_data = seurat_obj,
  phenotype = survival_data,
  phenotype_class = "survival",
  tirank_params = list(
    encoder_type = "Transformer",
    n_trials = 3L
  ),
  save_path = "./TiRank_survival_res"
)

# Access results
modified_seurat <- result$scRNA_data
head(modified_seurat[[]])

## End(Not run)
```

DownloadCellTypistModels

Download CellTypist Models

Description

Downloads CellTypist pre-trained models via the **celltypist** Python backend. A single model, multiple models, or all models can be downloaded. The models are saved to CellTypist's local models directory and can subsequently be used by `CellTypistAnnotate` without re-downloading each time.

To see all available models and their descriptions, see https://github.com/Teichlab/celltypist.

Usage

```
DownloadCellTypistModels(model = NULL, force_update = FALSE)
```

Arguments

model	Character. Name(s) of the CellTypist model(s) to download, e.g. "Immune_All_Low". A .pkl suffix is appended automatically if missing. When NULL, all models are downloaded. Default: NULL.
force_update	Logical. Whether to force re-downloading the CellTypist model files even when they already exist locally. Default: FALSE.

Value

Invisibly NULL. The models are downloaded to the CellTypist models directory (see models.models_path in the Python backend).

Requirements

- A Python environment with celltypist installed, discoverable by reticulate.

See Also

Other Single_Cell_Annotation_Method: CellTypistAnnotate(), ListCellTypistModels(), RegisterAnnoMethod(), SCAnnotateStrategy, SingleRAnnotate(), mLLMCelltypeAnnotate()

Examples

```
## Not run:
# Download all CellTypist models
DownloadCellTypistModels()

# Download a single model
DownloadCellTypistModels(model = "Immune_All_Low")

# Force re-downloading a specific model
DownloadCellTypistModels(model = "Immune_All_Low", force_update = TRUE)

## End(Not run)
```

DrawBulkPCA *Draw PCA Plot for Bulk Expression Data*

Description

Performs PCA on transposed bulk expression data and visualizes the first two principal components as a scatter plot with marginal density plots. Optionally overlays batch information as point shapes (up to 4 batches).

The plot is composed via patchwork from:

- A top marginal density plot for PC1.
- A right marginal density plot for PC2.
- A central scatter plot with group-colored ellipses (via ggforce::geom_mark_ellipse()).

Usage

```
DrawBulkPCA(bulk, group, batch = NULL, show_plot = TRUE, ...)
```

Arguments

bulk	A numeric matrix or data frame of bulk expression data (genes x samples). Column names are used as sample identifiers.
group	A vector of group labels, one per sample (column of bulk). Used for coloring points and ellipses.
batch	An optional vector of batch labels, one per sample. When provided, points are additionally shaped by batch (up to 4 batches supported). Default: NULL.
show_plot	Logical. If TRUE (default), the plot is printed immediately.
...	Additional arguments (currently unused).

Value

Invisibly returns the combined patchwork/ggplot object.

See Also

Other input_preprocess: PhenoMap(), PhenoPreProcess(), SCPreProcess(), SymbolConvert()

Examples

```
## Not run:
# Small example matrix: 10 genes x 8 samples
set.seed(123)
bulk <- matrix(
  rnorm(100 * 80, mean = 10, sd = 2),
  nrow = 100, ncol = 80,
  dimnames = list(
    paste0("Gene", 1:100),
    paste0("Sample", 1:80)
  )
)

group <- rep(c("Control", "Treatment"), each = 40)

# Basic PCA plot without batch
DrawBulkPCA(bulk = bulk, group = group)

# PCA plot with batch shapes
batch <- rep(c("Batch1", "Batch2"), times = 40)
DrawBulkPCA(bulk = bulk, group = group, batch = batch)

## End(Not run)
```

`FindRobustElbow` *Automatically determine optimal PCA dimensions using multiple robust methods*

Description

This function combines multiple statistical approaches to automatically determine the optimal number of principal components (PCs) for downstream single-cell analysis. It integrates variance-based heuristics, elbow detection algorithms, and provides comprehensive visualization for result validation.

Usage

```
FindRobustElbow(
  obj,
  verbose = SigBridgeRUtils::getFuncOption("verbose") %||% TRUE,
  ndims = 50L
)
```

Arguments

`obj`	A Seurat object that has PCA computed (after `RunPCA`) or a numeric matrix of cell embeddings (cells x PCs).
`verbose`	Logical, if TRUE outputs detailed method results and creates visualization plot. If `FALSE` returns only the final dimension.
`ndims`	Integer, maximum number of dimensions to consider (default: `50L`). Values exceeding the number of PCs stored in the `"pca"` reduction are capped at the available number of PCs.

Details

The numerical work is delegated to the C++ routine `find_robust_elbow_cpp()`, which receives the PCA cell embeddings as an S4-aware matrix view created by `beachmat::initializeCpp()`. Columns are streamed through `tatami` (see `Rtatami.h`), so the embeddings are never coerced into a dense base `matrix` and only the requested `ndims` columns are touched. For each of those PCs the routine computes the standard deviation, the explained variance, the percentage of explained variance, and the cumulative percentage, which are then fed into the six heuristics described below.

The recommended dimension is the second largest value among the following candidates:

- cumulative variance above 90\
- cumulative variance above 80\
- last PC explaining more variance than the mean;
- last PC explaining more than two standard deviations of the variance;
- maximum second derivative of the variance curve;
- largest distance between the variance curve and the chord joining the first and the last PC;

- the fixed anchors 10, 15, 20, 25, 30 and 35.

The fixed anchors are independent of the data, so the raw recommendation can exceed the number of PCs that were actually computed. The final value is therefore capped at the number of available PCs (i.e. `ndims`).

When `verbose = TRUE` and the session is interactive, the per-method candidates are plotted on top of the variance curve by the internal helper `plot_robust_elbow()`.

Value

Integer, the recommended number of PCA dimensions for downstream analysis

See Also

Other single_cell_preprocess: `FilterTumorCell()`, `Pattern2Colname()`, `QCPatternDetect()`, `RegisterSeuratMethod()`, `SCAnnotate()`, `SCIntegrate()`, `SCPreProcess()`, `SCPreProcessStrategy`

Examples

```
## Not run:
# After running PCA on Seurat object
pbmc <- RunPCA(pbmc, npcs = 50)
optimal_dims <- FindRobustElbow(pbmc, verbose = TRUE)
pbmc <- FindNeighbors(pbmc, dims = 1:optimal_dims)

## End(Not run)
```

FPKMToTPM *Convert bulk RNA-seq FPKM to TPM*

Description

Converts a gene-level FPKM matrix to TPM (Transcripts Per Million). Because the per-gene length factor is constant within a sample, the FPKM-to-TPM transformation requires no gene lengths: each column is scaled so that its values sum to 10^6.

For a given sample, every FPKM value is multiplied by $10^6 / \sum(\text{FPKM})$. Samples whose FPKM column sums to zero produce a zero column in the result and trigger a warning.

Missing values (`NA`) are either replaced by zero (`na_as_zero = TRUE`, the default) or trigger an error (`na_as_zero = FALSE`).

Usage

```
FPKMToTPM(data, na_as_zero = TRUE, verbose = TRUE, ...)
```

Arguments

data	A two-dimensional matrix-like object of FPKM values (a base matrix, a data frame, or an S4 Matrix). Rows are genes, columns are samples. Data frames are coerced and must contain only numeric columns. Values must be non-negative and finite.
na_as_zero	A logical flag. If TRUE (default), missing values are replaced by zero before normalization. If FALSE, an error is raised when data contains missing values.
verbose	A logical flag. If TRUE (default), diagnostic messages (missing-value report and completion summary) are shown.
...	Additional arguments passed to CheckNA() when data contains missing values and verbose is TRUE (for example max_print).

Value

A numeric TPM matrix with the same dimensions and dimnames as data. Each column sums to 10^6 unless the corresponding FPKM column sums to zero.

Examples

```
## Not run:
# Create an FPKM matrix (3 genes x 2 samples)
fpkm <- matrix(
  c(10, 20, 30, 15, 25, 35),
  nrow = 3, ncol = 2,
  dimnames = list(c("GENE1", "GENE2", "GENE3"), c("Sample1", "Sample2"))
)

# Convert FPKM to TPM
tpm <- FPKMToTPM(fpkm)
colSums(tpm)  # Each column should sum to 1e6

# Error if missing values are present and na_as_zero = FALSE
fpkm_na <- fpkm
fpkm_na[1, 1] <- NA
tpm2 <- FPKMToTPM(fpkm_na, na_as_zero = FALSE)

## End(Not run)
```

---

getFuncOption *Configuration Functions for SigBridgeR Package*

Description

These functions provide a centralized configuration system for the SigBridgeR package, allowing users to set and retrieve package-specific options with automatic naming conventions.

setFunOption sets one or more configuration options for the SigBridgeR package. Options are automatically prefixed with "SigBridgeR." if not already present, ensuring proper namespace isolation.

getFuncOption retrieves configuration options for the SigBridgeR package. The function automatically handles the "SigBridgeR." prefix, allowing users to reference options with or without the explicit prefix.

checkFuncOption validates configuration options for the SigBridgeR package. This function ensures that all options meet type and value requirements before they are set in the global options.

Usage

```
getFuncOption(option = NULL, default = NULL)
```

Arguments

`option`	Character string specifying the option name to check
`default`	The default value to return if the option is not set. Defaults to NULL.

Details

This function performs the following validations:

verbose Must be single logical values (TRUE/FALSE)

timeout, seed Must be single integer values

nthreads Must be a single positive integer value

The function is automatically called by setFuncOption to ensure all configuration options are valid before they are set.

Value

Invisibly returns NULL. The function is called for its side effects of setting package options.

The value of the specified option if it exists, otherwise the provided default value.

No return value. The function throws an error if the value doesn't meet the required specifications for the given option.

Available Opions

- `verbose`: A logical value indicating whether to print verbose messages, defaults to `TRUE`
- `timeout`: An integer specifying the timeout in seconds for parallel processing, defaults to '180L"
- `seed`: An integer specifying the random seed for reproducible results, defaults to '123L"
- `nthreads`: An integer specifying the number of threads for parallel processing, defaults to '1L"

Examples

```
# Set options with automatic prefixing
setFuncOption(verbose = TRUE)

# Retrieve options with automatic prefixing
getFuncOption("verbose")
```

`GetGeneLength` *Compute Gene Lengths from a GTF File or GTF Lines*

Description

An S3 generic that computes per-gene lengths by parsing exon features from either a GTF/GTF.GZ file or a character vector of GTF lines. The reported length for each gene is the total merged (non-overlapping) span of its exons.

The `character` method accepts two forms of input:

- A **single-element** character vector `x` is treated as a GTF file path (optionally gzip-compressed). The file is streamed line-by-line and is never loaded fully into memory.
- A **multi-element** character vector `x` is treated as the GTF lines themselves (e.g. the output of `readLines()`).

Overlapping or adjacent exon intervals are merged within each (`gene_id`, chromosome, strand) group, and the length of the merged span is reported. Ensembl `gene_id` version suffixes (e.g. `ENSG00000139618.15`) are preserved, so gene identifiers from different annotation builds remain distinct.

By default, exon records on **every** chromosome are kept. Set `canonical_chr_only = TRUE` to restrict the analysis to canonical chromosomes (autosomes `1-22` and chromosomes `X`, `Y`, and `MT`) and ignore any other contigs or scaffolds present in the annotation.

Usage

```
GetGeneLength(x, verbose = TRUE, canonical_chr_only = FALSE, ...)

## Default S3 method:
GetGeneLength(x, verbose = TRUE, canonical_chr_only = FALSE, ...)

## S3 method for class 'character'
GetGeneLength(x, verbose = TRUE, canonical_chr_only = FALSE, ...)
```

Arguments

`x`	The object to dispatch on: • A length-1 character string: path to a GTF or GTF.GZ file. • A character vector with more than one element: the GTF lines.

verbose — A logical flag. If TRUE (default), cli-style progress messages (with ANSI colors) are printed while reading, parsing, and merging the exon intervals.

canonical_chr_only — A logical flag. If TRUE, only exon records located on canonical chromosomes (autosomes 1-22, and chromosomes X, Y, MT) are retained; exons on any other contig/scaffold are ignored. If FALSE (default), exons on every chromosome are considered.

... — Reserved for future use.

Value

A named numeric vector of gene lengths, one entry per gene, with the gene ids as names. An empty (zero-length) named vector is returned when the input contains no valid exon records.

Examples

```
## Not run:
# Gene lengths from a gzip-compressed GTF file
lengths <- GetGeneLength("Homo_sapiens.GRCh37.75.gtf.gz")

# Gene lengths from GTF lines already read into R
x <- readLines("Homo_sapiens.GRCh37.75.gtf.gz")
lengths <- GetGeneLength(x)

## End(Not run)
```

InterceptStrategy *Inspect Registered Strategy Environments*

Description

Lists and optionally inspects the contents of predefined strategy environments (e.g., ScreenStrategy, SCPreProcessStrategy, AnnotationStrategy). Returns a summary of registered methods or variables, with optional detailed introspection of function signatures, body structure, and closure properties.

This utility aids in debugging, validation, and exploration of dynamically registered pipelines-particularly useful when developing or auditing extensible analysis frameworks.

Usage

```
InterceptStrategy(
  strategy = c("ScreenStrategy", "SCPreProcessStrategy", "SCAnnotateStrategy"),
  ...
)
```

Arguments

strategy	Character/Environment. strategy environment to inspect. Must be one of: "ScreenStrategy", "SCPreProcessStrategy", or "SCAnnotateStrategy". Partial matching is supported.
...	Additional arguments (currently unused, reserved for future extension).

Value

Invisibly returns a tibble

Examples

```
# List all preprocessing steps
InterceptStrategy("SCPreProcessStrategy")

# Get detailed function metadata for screen methods
details <- InterceptStrategy("ScreenStrategy")
```

IsCountsMatrix *Check whether a matrix looks like RNA-seq raw counts*

Description

Determines whether a numeric matrix is likely to be a bulk/single-cell RNA-seq raw counts matrix. A matrix is accepted only if it is non-empty, finite, non-negative, integer-like in a sufficiently high proportion, and has a positive library size for every sample. Useful for validating input before running count-based downstream analyses.

Usage

```
IsCountsMatrix(
  x,
  verbose = TRUE,
  integer_tol = 1e-08,
  min_integer_fraction = 0.95
)
```

Arguments

x	Numeric matrix. Rows are genes, columns are samples.
verbose	Logical. Whether to print diagnostic statistics (fractions of non-negative, integer-like, and zero values, library sizes, and the reason for rejection) to the console. Default: TRUE.
integer_tol	Numeric in [0, 1]. Absolute tolerance used to decide whether a value is integer-like (abs(x - round(x)) <= integer_tol). Default: 1e-8.

`min_integer_fraction`
: Numeric in `[0, 1]`. Minimum fraction of integer-like values required for the matrix to be considered counts. Default: `0.95`.

Details

`IsCountsMatrix()` rejects a matrix for any of the following reasons:

- the matrix is empty;
- the matrix contains `NA`, `NaN`, or infinite values;
- the matrix contains negative values;
- the fraction of integer-like values is below `min_integer_fraction`;
- at least one sample (column) has a library size (column sum) `<= 0`.

When `verbose = TRUE`, the reason for rejection (or diagnostic statistics on acceptance) is printed to the console.

Value

A single logical value: `TRUE` if `x` is likely a bulk counts matrix, `FALSE` otherwise.

Examples

```
# A typical bulk RNA-seq raw counts matrix (genes x samples)
counts <- matrix(
  c(100, 200, 300, 150, 250, 350),
  nrow = 3, ncol = 2,
  dimnames = list(c("GENE1", "GENE2", "GENE3"), c("Sample1", "Sample2"))
)
IsCountsMatrix(counts)

# A log-normalized matrix is not counts
IsCountsMatrix(log1p(counts))

# Matrices containing negative values are not counts
IsCountsMatrix(counts - 1000)

# Inspect diagnostic statistics
IsCountsMatrix(counts, verbose = TRUE)
```

`IsSkewedDynamic` *Test Whether the Proportion of a Target Value Is Highly Skewed*

Description

`IsSkewedDynamic()` tests whether the observed proportion of a specified target/background value in a vector deviates substantially from an expected proportion.

The function is designed for detecting highly skewed phenotype-like vectors, especially binary or discrete vectors where one value represents the background/control class. If the observed proportion of `target` differs from `expected_p` by more than `n_sd` standard deviations under a binomial model, the vector is considered skewed.

Formally, let:

$$\hat{p} = \frac{\#(x_i = target)}{n}$$

and

$$SD = \sqrt{\frac{expected_p(1 - expected_p)}{n}}$$

The function returns `TRUE` when:

$$|\hat{p} - expected_p| > n_sd \times SD$$

Otherwise, it returns `FALSE`.

If `x` has length zero, the function returns `NA`.

Usage

```
IsSkewedDynamic(x, target = 0L, expected_p = 0.8, n_sd = 4L)
```

Arguments

`x`	A vector to be tested. Must be numeric, integer, or logical. Missing values are not allowed.
`target`	A single numeric value indicating the target/background value whose proportion should be evaluated. Default is `0`.
`expected_p`	A single numeric value between 0 and 1 indicating the expected proportion of `target` in `x`. Default is `0.8`.
`n_sd`	A single integer or numeric value specifying how many standard deviations away from `expected_p` are allowed before the vector is considered skewed. Default is `4L`.

Details

This function assumes the expected occurrence of `target` follows a binomial proportion model with probability `expected_p`. The standard deviation is computed as:

$$\sqrt{expected_p(1 - expected_p)/n}$$

where n is the length of x.

The check is symmetric: both an unexpectedly high and unexpectedly low proportion of target are considered skewed.

Internally, the computation is delegated to a C++ implementation via Rcpp for efficiency.

Value

A logical vector of length 1:

- TRUE if the observed proportion of target deviates from expected_p by more than n_sd standard deviations.
- FALSE otherwise.
- NA if x has length zero.

Missing values

x must not contain missing values. If any NA is present, the function throws an error.

Supported input types

x may be one of:

- numeric
- integer
- logical

Other input types are rejected by the underlying C++ implementation.

Examples

```
x1 <- c(rep(0, 80), rep(1, 20))
IsSkewedDynamic(x1)

x2 <- c(rep(0, 95), rep(1, 5))
IsSkewedDynamic(x2)

x3 <- c(rep(0, 50), rep(1, 50))
IsSkewedDynamic(x3, expected_p = 0.8)

IsSkewedDynamic(numeric(0))
```

`list_to_chr` *Convert a List to a Compact Character Representation*

Description

Converts an R `list` into a single character string using the format `"key: value"`. Named elements use their names as keys, while unnamed elements use their 1-based positions. Nested lists are converted recursively and enclosed in parentheses. This method is registered as `list_to_chr`, so it is called automatically when `as.character()` is applied to a list.

Usage

```
list_to_chr(x)
```

Arguments

`x`	A list to be converted.

Value

A character vector of length one.

Examples

```
x <- list(key1 = "val1", key2 = "val2")
as.character(x)
# "key1: val1, key2: val2"
x <- list(key1 = list(key2 = list(key3 = "val3")))
as.character(x)
# "key1: (key2: (key3: val3))"
x <- list(
  a = 1,
  b = list(
    c = 2,
    d = list(e = "hello")
  ),
  f = c(1, 2, 3)
)
as.character(x)
# "a: 1, b: (c: 2, d: (e: hello)), f: 1, 2, 3"
x <- list("foo", bar = NA, baz = NULL)
as.character(x)
# "1: foo, bar: NA, baz: NULL"
```

`list_to_chr_formated` *Convert a List to an Indented Multi-line Character Representation*

Description

Converts an R `list` into a single character string in which every element occupies its own line, using the format `"key: value"`. Named elements use their names as keys, while unnamed elements use their 1-based positions. Nested lists are expanded recursively, and each nesting level is indented by two additional spaces.

Usage

```
list_to_chr_formated(x)
```

Arguments

`x`	A list to be converted.

Value

A character vector of length one, with lines separated by `"\n"` (no trailing newline).

Examples

```
x <- list(
  a = 1,
  b = list(
    c = 2,
    d = list(e = "hello")
  ),
  f = c(1, 2, 3)
)
cat(list_to_chr_formated(x), "\n")
# a: 1
# b:
#   c: 2
#   d:
#     e: hello
# f: 1, 2, 3
```

`ListCellTypistModels` *List Available CellTypist Models*

Description

Lists the CellTypist pre-trained models available locally. By default it reads all model files under `~/.celltypist/data/models`, the default CellTypist models directory (overridable via the `CELLTYPIST_FOLDER` environment variable). Models are downloaded with `DownloadCellTypistModels` and used by `CellTypistAnnotate`.

Usage

```
ListCellTypistModels(path = NULL, verbose = getFuncOption("verbose"))
```

Arguments

`path`	Character. Path to the CellTypist models directory. Default: `NULL`, which resolves to `$CELLTYPIST_FOLDER/data/models` (or `~/.celltypist/data/models` when the environment variable is unset).
`verbose`	Logical. Whether to print the model list to the console. Default: `getFuncOption("verbose")`.

Value

A character vector of available model names (without the `.pkl` suffix). Invisibly returned when `verbose = TRUE`.

See Also

Other Single_Cell_Annotation_Method: `CellTypistAnnotate()`, `DownloadCellTypistModels()`, `RegisterAnnoMethod()`, `SCAnnotateStrategy`, `SingleRAnnotate()`, `mLLMCelltypeAnnotate()`

Examples

```
## Not run:
# List all locally available CellTypist models
ListCellTypistModels()

# List models in a custom directory, without printing
ListCellTypistModels(path = "~/my_models", verbose = FALSE)

## End(Not run)
```

`ListPyEnv` *List Available Python Environments*

Description

Discovers and lists available Python environments of various types on the system. This generic function provides a unified interface to find Conda environments and virtual environments (venv) through S3 method dispatch.

Default method that lists all Python environments by combining results from Conda and virtual environment discovery methods.

Discovers Conda environments using multiple detection strategies for maximum reliability. First attempts to use system Conda commands, then falls back to reticulate's built-in Conda interface if Conda command is unavailable or fails. Returns empty data frame if Conda is not available or no environments are found.

Discovers virtual environments by searching common venv locations including user directories (`~/.virtualenvs`, `~/.venvs`) and project folders (`./venv`, `./.venv`). Supports custom search paths through the `venv_locations` parameter. Returns empty data frame if no virtual environments are found in the specified locations.

Usage

```
ListPyEnv(
  env_type = c("all", "conda", "venv", "virtualenv"),
  timeout = 30000L,
  venv_locations = c("~/.virtualenvs", "~/.venvs", "./venv", "./.venv"),
  verbose = TRUE,
  ...
)
```

Arguments

`env_type`	Character string specifying the type of environments to list. One of: `"all"`, `"conda"`, `"venv"`. Defaults to `"all"`.
`timeout`	The maximum timeout time when using system commands, only effective when `env_type=conda`.
`venv_locations`	Character vector of directory paths to search for virtual environments. Default includes standard locations and common project directories.
`verbose`	Logical indicating whether to print verbose output.
`...`	For future use.

Details

The function uses S3 method dispatch to handle different environment types:

- `"all"`: Combines results from all environment types using `rbind()`

- "conda": Searches for Conda environments using multiple methods:
  - Primary: reticulate::conda_list() for reliable environment detection
  - Fallback: System conda info --envs command for broader compatibility
- "venv": Searches common virtual environment locations including user directories and project folders

Each method includes comprehensive error handling and will return empty results with informative warnings if no environments are found or if errors occur during discovery.

Value

A data frame with the following columns:

- name - Character vector of environment names
- python - Character vector of paths to Python executables
- type - Character vector indicating environment type ("conda" or "venv")

Returns an empty data frame with these columns if no environments are found.

Examples

```
## Not run:
# List all Python environments
ListPyEnv("all")

# List only Conda environments
ListPyEnv("conda")

# List only virtual environments with custom search paths
ListPyEnv("venv", venv_locations = c("~/my_envs", "./project_env"))

## End(Not run)
```

LoadCache *Load a Cached R Object*

Description

Reads a cached R object from disk. The file format is determined by the file extension:

- .qs2: loaded via qs2::qs_read()
- .RData / .rds: loaded via readRDS()
- .csv: loaded via data.table::fread()

If the file was originally saved in .qs2 format but qs2 is not installed on the current machine, a clear error message is shown with installation instructions.

Usage

```
LoadCache(file)
```

Arguments

file	Character string. Path to the cache file to load.

Value

The cached R object.

See Also

Other cache_config: CacheSetHere(), CacheSysCall(), CheckCache(), ChooseCache(), WriteCache(), WriteCacheMeta()

Examples

```
## Not run:
result <- LoadCache("Scissor_res/survival_2025_01_01/result.qs2")
feature_df <- LoadCache("Scissor_res/survival_2025_01_01/feature_table.csv")

## End(Not run)
```

LoadRefData *Download & Load Reference Data*

Description

This function checks if the data already exists in a local cache. If not, it downloads the file from the remote repository using multiple sources with fallback.

Usage

```
LoadRefData(
  data_type = c("survival", "binary", "continuous"),
  path = NULL,
  timeout = SigBridgeRUtils::getFuncOption("timeout"),
  ...
)
```

Arguments

data_type	The type of data to download. Must be one of "survival", "binary", or "continuous".
path	Optional path to save the downloaded file, default: NULL, saving in package.
timeout	Integer. Connection timeout in seconds (default: 60)

... Additional arguments. Currently supports:

- verbose: Logical indicating whether to print progress messages. Defaults to TRUE.

Value

The requested datasets, stored in a list.

Examples

```
## Not run:
ref_data <- LoadRefData(data_type = c("survival"))

## End(Not run)
```

MergeResult *Merge Multiple Screening Analysis Results*

Description

Combines results from multiple single-cell screening analyses (e.g., Scissor, scPAS, scPP, scAB) by merging their metadata, slots, and miscellaneous information while preserving the original expression data from the first input. Performs an inner join on cell barcodes to ensure only cells present in all inputs are retained.

Usage

```
MergeResult(
  ...,
  weights = NULL,
  ties.method = c("random", "first", "last"),
  verbose = SigBridgeRUtils::getFuncOption("verbose")
)
```

Arguments

... Input objects to merge. Can be:

- Seurat objects
- Lists containing scRNA_data (Seurat objects)
- Mixed combinations of the above

The first object serves as the base for merging. When duplicate metadata columns are found, priority is given to the first occurrence.

weights Named numeric vector specifying weights for voting when multiple screening methods produce categorical labels. Names must match screening method names (case-insensitive). If NULL (default), no voting is performed.

`ties.method` Character. Method for resolving ties in weighted voting:

- `"first"`: Select the first category alphabetically (default)
- `"last"`: Select the last category alphabetically
- `"random"`: Randomly select among tied categories

`verbose` Logical. Whether to print progress messages. Default: inherits from `getOption("SigBridgeR.verbose'`

Value

A merged Seurat object containing:

- Expression data from the first input object
- Combined metadata from all input objects (inner join on cell IDs)
- Merged slots (assays, reductions, graphs, images) from all objects
- Combined miscellaneous information from all objects
- Optional `vote` column in metadata if `weight` is provided

Processing Workflow

1. **Input Validation**: Extracts Seurat objects from inputs (skips invalid objects with warning)
2. **Metadata Extraction**: Converts metadata to data.table format for efficient merging
3. **Cell Intersection**: Retains only cells present in all datasets (inner join)
4. **Weighted Voting**: If `weight` provided, performs weighted voting across screening methods
5. **Slot Merging**: Combines assays, reductions, graphs, and images from all objects
6. **Misc Merging**: Aggregates miscellaneous information from all objects

Weighted Voting

When `weight` is provided, the function:

1. Identifies columns in metadata matching registered screening method names
2. Extracts vote data (must contain "Positive", "Negative", "Neutral", or "Other")
3. Applies `WeightedVote` with specified weights
4. Adds `vote` column to merged metadata with final aggregated labels

Notes

- Only cells present in *all* input objects are retained (inner join behavior)
- Duplicate metadata columns are resolved by keeping the first occurrence
- For integrating *heterogeneous* single-cell datasets (different samples), use `SCIntegrate` instead
- Weights should be non-negative; higher values indicate greater influence in voting

See Also

`WeightedVote`, `SCIntegrate`, `AddMisc`

Examples

```
## Not run:
# Example 1: Merge results from different screening methods
scissor_out <- RunScissor(...)
scAB_out <- RunscAB(...)
scPP_out <- RunscPP(...)

merged <- MergeResult(scissor_out, scAB_out, scPP_out)

# Check merged metadata
head(merged[[]])

# Example 2: Weighted voting across methods
weights <- c(
  Scissor = 3,    # Highest confidence
  scAB = 2,       # Medium confidence
  scPP = 1        # Lower confidence
)

merged <- MergeResult(
  scissor_out,
  scAB_out,
  scPP_out,
  weight = weights,
  ties.method = "first"
)

# View voting results
table(merged$vote)

# Example 3: Merge list-containing objects
result_list1 <- list(scRNA_data = seurat_obj1, other_info = "...")
result_list2 <- list(scRNA_data = seurat_obj2, other_info = "...")

merged <- MergeResult(result_list1, result_list2)

# Example 4: Check which cells were retained
original_cells <- ncol(seurat_obj1)
merged_cells <- ncol(merged)
cat("Retained", merged_cells, "of", original_cells, "cells\n")

# Example 5: Access merged slots
# All unique assays from all inputs
names(merged@assays)

# All unique reductions
names(merged@reductions)

## End(Not run)
```

`Method` *Method: Abstract Base Class for Analysis Methods*

Description

`Method` is the abstract base class for all analysis methods (screening and annotation) in the SigBridgeR package. It defines the common interface that every method must implement, including a method name, version, and executor function.

Usage

```
Method(
  method_name,
  executor,
  method_version,
  sigbridger_version = get_pkg_version()
)
```

Arguments

`method_name`	`character`. A string identifying the method (e.g., `"Scissor"`, `"scPAS"`).
`executor`	`function`. The function that performs the actual computation. This is the core workhorse of each method.
`method_version`	`character`. A string indicating the version of the underlying software package.
`sigbridger_version`	Package Version of SigBridgeR

Details

This class is defined using the S7 object-oriented system and inherits from SigBridgeRBase. It is marked as abstract, meaning it cannot be instantiated directly. Concrete subclasses such as ScreenMethod and `AnnotationMethod` provide specific method categories.

Value

A `Method` S7 object (abstract; cannot be instantiated directly).

See Also

Other SigBridgeR-Classes: `ScreenMethod()`, `SigBridgeRBase()`

`MethodParamSet` *A List of Method Parameters*

Description

Collects the method-specific arguments of a benchmark configuration. Each method registered in ScreenStrategy has a corresponding argument named `"<method>_param"`, which defaults to `list()`.

Usage

```
MethodParamSet(
  Scissor_param = list(),
  scAB_param = list(),
  scPAS_param = list(),
  scPP_param = list(),
  DEGAS_param = list(),
  DEGASv2_param = list(),
  LP_SGL_param = list(),
  PIPET_param = list(),
  SCIPAC_param = list(),
  SIDISH_param = list(),
  TiRank_param = list(),
  PhenoMapR_param = list(),
  ...
)
```

Arguments

`Scissor_param`	Extra arguments for method Scissor
`scAB_param`	Extra arguments for method scAB
`scPAS_param`	Extra arguments for method scPAS
`scPP_param`	Extra arguments for method scPP
`DEGAS_param`	Extra arguments for method DEGAS
`DEGASv2_param`	Extra arguments for method DEGASv2
`LP_SGL_param`	Extra arguments for method LP_SGL
`PIPET_param`	Extra arguments for method PIPET
`SCIPAC_param`	Extra arguments for method SCIPAC
`SIDISH_param`	Extra arguments for method SIDISH
`TiRank_param`	Extra arguments for method TiRank
`PhenoMapR_param`	Extra arguments for method PhenoMapR
`...`	Currently unused

Value

A modified list of method-specific parameters.

See Also

Other ConfigSet: BenchGSEAConfigSet(), BenchGeneScoreConfigSet(), BenchMemoryConfigSet()

Examples

```
MethodParamSet(
  Scissor_param = list(alpha = 0.05),
  scPAS_param = list(alpha = 0.05)
)
```

---

mLLMCelltypeAnnotate *Annotate Cell Types Using Multi-LLM Consensus Approach*

Description

Performs automated cell type annotation for Seurat clusters by leveraging multiple large language models (LLMs) to interpret cluster-specific marker genes. The function supports both single-model annotation and multi-model consensus generation, with uncertainty quantification via agreement metrics.

Workflow:

1. Identifies top marker genes per cluster (via `Seurat::FindAllMarkers()` or user-provided markers).
2. Queries specified LLMs with marker gene lists and tissue context.
3. For multiple models: computes consensus annotations and uncertainty metrics (consensus proportion, entropy) via `mLLMCelltype::interactive_consensus_annotation()`.
4. For single model: uses direct annotation via `mLLMCelltype::annotate_cell_types()`.
5. Adds results as metadata columns to the Seurat object.

Usage

```
mLLMCelltypeAnnotate(sc, ...)

## Default S3 method:
mLLMCelltypeAnnotate(sc, ...)

## S3 method for class '`SigBridgeR::ScreenMethodResult`'
mLLMCelltypeAnnotate(sc, ...)

## S3 method for class 'SingleCellExperiment'
mLLMCelltypeAnnotate(sc, ...)
```

```
## S3 method for class 'Seurat'
mLLMCelltypeAnnotate(
  sc,
  seurat_obj_markers = NULL,
  tissue_name = NULL,
  models = vector("character", 0L),
  api_keys = list(),
  top_gene_count = 10L,
  debug = FALSE,
  base_urls = NULL,
  return_reasoning = FALSE,
  controversy_threshold = 0.7,
  entropy_threshold = 1L,
  max_discussion_rounds = 3L,
  consensus_check_model = NULL,
  log_dir = "logs",
  cache_dir = NULL,
  use_cache = TRUE,
  clusters_to_analyze = NULL,
  force_rerun = FALSE,
  ...
)
```

Arguments

sc	A single-cell dataset with pre-computed clusters (stored in `Idents(sc)` or `sc$seurat_clusters`), one of: • A `Seurat` object (recommended); • A `SingleCellExperiment` object, converted to `Seurat` via `Seurat::as.Seurat()` before annotation (requires `counts`/`logcounts` assays); • A ScreenMethodResult object, whose `scRNA_data` slot is annotated.
...	Additional arguments routed to downstream functions: • Arguments matching `Seurat::FindAllMarkers()` formals (e.g., `min.pct`, `logfc.threshold`, `test.use`) are passed to marker detection when `seurat_obj_markers = NULL`. • `verbose`: Logical. Whether to print progress messages. Default: `getFuncOption("verbose")`. • `seed`: Integer. Random seed. Default: `getFuncOption("seed")`.
seurat_obj_markers	Optional pre-computed marker gene table (output of `Seurat::FindAllMarkers()`). If `NULL` (default), markers are computed automatically using parameters passed via `...`.
tissue_name	Character. Biological context for annotation (e.g., tissue type or disease state), which helps LLMs interpret marker genes appropriately. Default: `NULL`.
models	Character vector of LLM model identifiers. Supported formats: • OpenAI: `"gpt-4o"`, `"gpt-4o-mini"`, etc.

- Anthropic: "claude-3-5-sonnet-20240620", etc.
- Google: "gemini-1.5-pro", etc.
- Alibaba: "qwen-max", "qwen-plus", etc.

For single-model mode (length(models) == 1), only the first model is used via mLLMCelltype::annotate_cell_types(). Must be provided. Default: character().

api_keys Named vector of API keys with provider names as keys:

- openai: OpenAI API key
- anthropic: Anthropic API key
- gemini: Google Cloud API key (with Gemini enabled)
- qwen: Alibaba DashScope API key

Example: list(openai = "sk-...", anthropic = "sk-ant-..."). Must be provided and named; placeholder keys (e.g., "your-xxx-key") will fail. Default: list().

top_gene_count Integer. Number of top marker genes per cluster passed to the LLMs. Default: 10.

debug Logical. Whether to enable debug output from mLLMCelltype::annotate_cell_types(). Default: FALSE.

base_urls Character vector or list of custom base URLs for the LLM API endpoints (e.g., for proxies or local deployments). Default: NULL (use the default endpoints).

return_reasoning
Logical. Whether to return the reasoning text from the single-model annotation. Default: FALSE.

controversy_threshold
Numeric. Consensus proportion below which a cluster is considered controversial and triggers further discussion. Default: 0.7.

entropy_threshold
Numeric. Entropy below which consensus is considered reached. Default: 1.

max_discussion_rounds
Integer. Maximum number of discussion rounds in the consensus process. Default: 3.

consensus_check_model
Character. Model used to check consensus during the discussion. Default: NULL (use the first model).

log_dir Character. Directory for saving the consensus process logs. Default: "logs".

cache_dir Character. Directory for caching API responses. Default: NULL.

use_cache Logical. Whether to use cached API responses. Default: TRUE.

clusters_to_analyze
Character or integer vector. Subset of cluster IDs to analyze; NULL analyzes all clusters. Default: NULL.

force_rerun Logical. Whether to ignore cached results and re-run the annotation. Default: FALSE.

Value

The input `Seurat` object with the following metadata columns added:

- `mllmcelltype_cell_type`: Consensus cell type annotation per cell.
- `mllmcelltype_consensus_proportion`: (Multi-model only) Proportion of models agreeing on the assigned label (range: 0-1). Higher values indicate stronger consensus.
- `mllmcelltype_entropy`: (Multi-model only) Shannon entropy of model predictions. Lower values indicate higher confidence (less disagreement among models).

Note: Uncertainty metrics are *only added in multi-model mode* (`length(models) > 1`).

`SingleCellExperiment` and ScreenMethodResult inputs are annotated as `Seurat` objects and returned as such.

Requirements

- R packages: `mLLMCelltype`, `Seurat`
- Valid API keys for the selected LLM providers (costs may apply)
- Internet connectivity for LLM API calls

See Also

`annotate_cell_types`, `interactive_consensus_annotation`

Other Single_Cell_Annotation_Method: `CellTypistAnnotate()`, `DownloadCellTypistModels()`, `ListCellTypistModels()`, `RegisterAnnoMethod()`, `SCAnnotateStrategy`, `SingleRAnnotate()`

Examples

```
## Not run:
# Multi-model consensus annotation
annotated <- mLLMCelltypeAnnotate(
  sc = pbmc_small,
  tissue_name = "Peripheral Blood Mononuclear Cells",
  models = c("gpt-4o", "claude-3-5-sonnet-20240620"),
  api_keys = list(
    openai = Sys.getenv("OPENAI_API_KEY"),
    anthropic = Sys.getenv("ANTHROPIC_API_KEY")
  ),
  top_gene_count = 15,
  min.pct = 0.25,
  logfc.threshold = 0.5
)

# User-provided markers
markers_list <- list(
  "0" = c("CD3D", "CD3E", "CD2", "IL7R", "LTB"),
  "1" = c("CD14", "LYZ", "CST3", "MS4A7", "FCGR3A")
)
# Example marker data frame
markers_df <- data.frame(
```

```
  cluster = c(0, 0, 0, 1, 1, 1),
  gene = c("CD3D", "CD3E", "CD2", "CD14", "LYZ", "CST3"),
  avg_log2FC = c(2.5, 2.3, 2.1, 3.1, 2.8, 2.5),
  p_val_adj = c(0.001, 0.001, 0.002, 0.0001, 0.0002, 0.0005)
)

# Single-model annotation (faster, no consensus metrics)
annotated <- mLLMCelltypeAnnotate(
  sc = pbmc_small,
  models = "gpt-4o-mini",
  api_keys = list(openai = Sys.getenv("OPENAI_API_KEY"))
)

# Inspect results
table(annotated$mllmcelltype_cell_type)
head(annotated$mllmcelltype_consensus_proportion)  # Only present in multi-model mode

## End(Not run)
```

`palette_SigBridgeR` *A Palette*

Description

A Palette

Usage

```
palette_SigBridgeR()
```

`Pattern2Colname` *convert regex patterns to column names (internal)*

Description

An internal utility function that converts regular expression patterns used for quality control in single-cell RNA-seq analysis into standardized column names for Seurat object metadata. This function handles both single patterns and combined patterns with logical OR operators.

Usage

```
Pattern2Colname(pat)
```

Arguments

`pat`	A character string containing a regular expression pattern or multiple patterns combined with | (OR operator).

Value

A character string with the standardized column name derived from the input pattern(s). The output follows these conventions: - Lowercase letters only - Special characters removed or replaced with underscores - Common patterns mapped to standardized abbreviations - Combined patterns sorted alphabetically and joined with underscores

See Also

Other single_cell_preprocess: `FilterTumorCell()`, `FindRobustElbow()`, `QCPatternDetect()`, `RegisterSeuratMethod()`, `SCAnnotate()`, `SCIntegrate()`, `SCPreProcess()`, `SCPreProcessStrategy`

Examples

```
## Not run:
# Internal usage examples
Pattern2Colname("^MT-")                    # Returns "mt"
Pattern2Colname("^RP[LS]")                 # Returns "rp"
Pattern2Colname("^[rt]rna")                # Returns "rt_rna"
Pattern2Colname("^MT-|^RP[LS]")            # Returns "mt_rp"
Pattern2Colname("^HB[AB]?")                # Returns "hb_ab"
Pattern2Colname("Custom_Pattern[0-9]+")    # Returns "custom_pattern_0_9"

## End(Not run)
```

PhenoMap *Map Phenotype Values Using Conditional Rules*

Description

Efficiently transforms phenotype values based on user-defined conditional rules. Similar to `dplyr::case_when` but optimized for performance on large datasets.

Usage

```
PhenoMap(data, ..., .default = NA)
```

Arguments

`data`	A vector, data frame, or data.table containing the phenotype data to transform.
`...`	Named or unnamed formulas of the form `condition ~ value`. Conditions are evaluated sequentially; the first matching condition determines the output value. Example: `col > 10 ~ "High", col <= 10 ~ "Low"`
`.default`	Value to use when no conditions match. Default: `NA`.

Value

A data.table with the transformed column. If input is a vector, returns a vector.

See Also

fcase, case_when

Other input_preprocess: DrawBulkPCA(), PhenoPreProcess(), SCPreProcess(), SymbolConvert()

Examples

```
## Not run:
# Example 1: Discretize a continuous phenotype
scores <- rnorm(100, mean = 50, sd = 10)
scores <- PhenoMap(scores, scores > 60 ~ "High", scores > 40 ~ "Medium", .default = "Low")

# Example 2: Transform a column in a data frame
df <- data.frame(
  age = c(25, 35, 45, 55, 65),
  gender = c("M", "F", "M", "F", "M")
)
df <- PhenoMap(df, age < 30 ~ "Young", age < 50 ~ "Middle", .default = "Senior")

# Example 3: Multiple conditions with complex logic
df <- data.frame(value = c(5, 15, 25, 35, 45))
df <- PhenoMap(
  df,
  value < 10 ~ "Very Low",
  value < 20 ~ "Low",
  value < 30 ~ "Medium",
  value < 40 ~ "High",
  .default = "Very High"
)

## End(Not run)
```

PhenoPreProcess *Preprocess Phenotype Data*

Description

[Deprecated] Aligns and validates phenotype data with bulk RNA-seq expression matrix. Supports three phenotype types (binary, continuous, survival) with automatic sample matching, type validation, and optional conditional mapping via PhenoMap.

Usage

```
PhenoPreProcess(
  bulk,
  phenotype,
  phenotype_class = c("binary", "continuous", "survival"),
  ...,
```

```
  select = NULL,
  verbose = getFuncOption("verbose")
)
```

Arguments

bulk	A two-dimensional matrix or data frame of bulk RNA-seq expression data with genes as rows and samples as columns. Must have at least 2 samples.
phenotype	Phenotype data, either: • Named numeric vector (for binary/continuous phenotypes) • Data frame/matrix with row names matching `colnames(bulk)` (for survival or multi-column phenotypes)
phenotype_class	Character. Type of phenotype: • `"binary"`: Two-class categorical outcome (e.g., case/control). Must have exactly 2 unique values. • `"continuous"`: Continuous measurement (e.g., age, expression). Must have >2 unique values. • `"survival"`: Survival data with time and status columns. Automatically guesses columns named "time" and "status/censor" if `select` is `NULL`. Partial matching is supported.
...	Conditional mapping rules passed to PhenoMap for transforming phenotype values before validation. Format: `condition ~ value`. Example: `status == "death" ~ 1, status == "alive" ~ 0`.
select	Character. Column name(s) to select from phenotype when it is two-dimensional. Required for survival data with multiple columns, or to specify which column to use for binary/continuous phenotypes.
verbose	Logical. Whether to print diagnostic messages. Default: inherits from `getOption("SigBridgeRUtils.v`

Value

Preprocessed phenotype data:

- For binary/continuous: Named numeric vector with sample names as names
- For survival: Data frame with two columns (time, status) and sample names as row names

Only samples present in both bulk and phenotype are retained.

Validation Rules

- **Binary**: Must have exactly 2 unique values (e.g., 0/1, TRUE/FALSE)
- **Continuous**: Must have >2 unique values (to avoid confusion with binary)
- **Survival**: Must have exactly 2 unique values in status column (typically 0 = censored, 1 = event)
- **Sample matching**: Common samples identified via `intersect(colnames(bulk), names/rownames(phenotype))`

Automatic Features

- **Survival column guessing**: When `phenotype_class = "survival"` and `select = NULL`, automatically detects columns containing "time" and "status"/"censor" in their names (case-insensitive).
- **Type conversion**: Logical values are converted to numeric (TRUE->1, FALSE->0)
- **Sample alignment**: Returns only samples present in both `bulk` and `phenotype`

See Also

`PhenoMap()`

Other input_preprocess: `DrawBulkPCA()`, `PhenoMap()`, `SCPreProcess()`, `SymbolConvert()`

Examples

```
## Not run:
# Example 1: Binary phenotype
bulk_data <- matrix(rpois(100, 10), nrow = 20, ncol = 5)
colnames(bulk_data) <- paste0("Sample", 1:5)

pheno_binary <- c(Sample1 = 1, Sample2 = 0, Sample3 = 1, Sample4 = 0, Sample5 = 1)

result <- PhenoPreProcess(
  bulk = bulk_data,
  phenotype = pheno_binary,
  phenotype_class = "binary"
)

# Example 2: Continuous phenotype with discretization
pheno_age <- c(Sample1 = 25, Sample2 = 35, Sample3 = 45, Sample4 = 55, Sample5 = 65)

result <- PhenoPreProcess(
  bulk = bulk_data,
  phenotype = pheno_age,
  phenotype_class = "continuous",
  .x < 40 ~ "Young", .x >= 40 ~ "Old"
)

# Example 3: Survival data with automatic column detection
pheno_surv <- data.frame(
  time = c(12, 24, 18, 36, 30),
  status = c(1, 0, 1, 1, 0),
  row.names = paste0("Sample", 1:5)
)

result <- PhenoPreProcess(
  bulk = bulk_data,
  phenotype = pheno_surv,
  phenotype_class = "survival"
)

# Example 4: Survival data with explicit column names
```

```
pheno_surv_custom <- data.frame(
  follow_up = c(12, 24, 18, 36, 30),
  event = c(1, 0, 1, 1, 0),
  row.names = paste0("Sample", 1:5)
)

result <- PhenoPreProcess(
  bulk = bulk_data,
  phenotype = pheno_surv_custom,
  phenotype_class = "survival",
  select = c("follow_up", "event")
)

## End(Not run)
```

`plus-methods` *Merge two objects with* `+`

Description

`x + y` combines two screening results into a single Seurat object via `MergeResult()`. Both operands may be ScreenMethodResult objects or Seurat objects; a ScreenMethodResult is unwrapped to its `scRNA_data` slot before merging.

Usage

```
## S3 method for class '`SigBridgeR::ScreenMethodResult`'
x + y

## S3 method for class 'Seurat'
x + y
```

Arguments

`x`	A ScreenMethodResult or Seurat object as the main receiver.
`y`	A ScreenMethodResult or Seurat object as the main donor.

Value

A merged Seurat object.

Property-visitor *Retrieve a property from SigBridgeR object*

Description

Retrieve a property from SigBridgeR object

Usage

```
## S3 method for class '`SigBridgeR::SigBridgeRBase`'
object$name

## S3 method for class '`SigBridgeR::SigBridgeRBase`'
object[[name]]
```

Arguments

object	An object inheriting from SigBridgeRBase
name	The name of the parameter as a character. Partial matching is not performed.

QCFilter *Filter Seurat object cells by QC metrics*

Description

Filters cells based on nFeature_RNA and optional QC metrics (e.g. percent.mt, percent.rp), defined in `seurat_obj@misc$qc_colnames` (See QCPatternDetect). Only metrics with non-constant, non-all-zero values are used.

Usage

```
QCFilter(
  seurat_obj,
  data_filter.thresh = list(assay = SeuratObject::DefaultAssay(seurat_obj),
   nFeature_thresh = c(200L, 6000L), nCount_thresh = c(500L, 50000L), percent.mt = 20L,
    percent.rp = 60L),
  verbose = TRUE,
  ...
)
```

Arguments

seurat_obj	A Seurat object.
data_filter.thresh	A named list with thresholds. Default: list(nFeature_thresh = c(200L, 6000L), nCount_thresh = c(500L, 25000L), percent.mt = 20L, percent.rp = 60L).
verbose	Logical; whether to print progress messages via cli.
...	No use

Value

A filtered Seurat object.

QCPatternDetect *Calculate Percentage of Features Matching Patterns*

Description

This function calculates the percentage of counts coming from features matching specified patterns (e.g., mitochondrial genes, ribosomal genes) and adds them as metadata columns to the Seurat object.

Usage

```
QCPatternDetect(
  obj,
  pattern = c("^MT-", "^mt-", "^RP[SL]", "^MT-|^RP[SL]"),
  verbose = TRUE,
  ...
)
```

Arguments

obj	A seurat object.
pattern	A character vector or list containing regex patterns to identify mitochondrial genes, ribosomal protein genes, or other unwanted genes, as well as combinations of these genes. Customized patterns are supported.
verbose	logical, whether to print progress messages
...	Additional arguments passed to PercentageFeatureSet

Details

The function automatically generates friendly column names based on the patterns:

- "mt" for mitochondrial patterns
- "rp" for ribosomal patterns
- "rrna" for ribosomal RNA patterns
- For combined patterns (using |), creates names like "mt_rp"
- For other patterns, creates cleaned lowercase names

See Also

Other single_cell_preprocess: `FilterTumorCell()`, `FindRobustElbow()`, `Pattern2Colname()`, `RegisterSeuratMethod()`, `SCAnnotate()`, `SCIntegrate()`, `SCPreProcess()`, `SCPreProcessStrategy`

`Register` *Unified Registration Interface for Strategy Methods*

Description

Registers screening, preprocessing, or annotation methods into their respective strategy registries. `Register()` accepts named arguments where the name becomes the registry key and the value (an S7 object or function) is dispatched to the appropriate `RegisterImpl` method.

Usage

```
Register(..., overwrite = FALSE, verbose = getFuncOption("verbose"))
```

Arguments

`...`	Named arguments where each name is the registry key and each value is an S7 object (ScreenMethod or `AnnotationMethod`) or a function (for `SCPreProcessStrategy`).
`overwrite`	`logical`. Whether to overwrite an existing entry with the same name. Default: `FALSE`.
`verbose`	`logical`. Whether to print registration success messages. Default: `getFuncOption("verbose")`.

Details

`Register()` iterates over its named `...` arguments, calling `RegisterImpl()` for each. `RegisterImpl` is an S7 generic that dispatches based on the class of the value:

- ScreenMethod -> registered into ScreenStrategy.
- AnnotationMethod -> registered into SCAnnotateStrategy.
- `function` -> validated as a Seurat/SeuratObject function and registered into SCPreProcessStrategy.

If the method name already exists in the target registry, the function aborts unless `overwrite = TRUE`.

Value

Invisibly TRUE on successful registration.

See Also

ScreenStrategy, SCAnnotateStrategy, SCPreProcessStrategy

Examples

```
## Not run:
# Register a ScreenMethod
method <- ScreenMethod(
  method_name = "Scissor",
  executor = DoScissor
)
Register(Scissor = method)

# Register a Seurat preprocessing function
Register(h = Seurat::RunHarmony)

# Register with overwrite
Register(Scissor = method, overwrite = TRUE)

## End(Not run)
```

RegisterAnnoMethod *Register an Annotation Method into the Strategy Registry*

Description

[Deprecated] Registers one or more user-defined annotation functions into a shared registry environment (e.g., SCAnnotateStrategy). Each method is stored with minimal metadata (method_name and executor) to enable dynamic dispatch in annotation pipelines.

This function supports extensible cell type annotation frameworks-ideal for integrating novel classifiers (e.g., custom SingleR wrappers, marker-based assigners, or deep learning models) while maintaining a uniform execution interface.

Usage

```
RegisterAnnoMethod(
  ...,
  registry = SCAnnotateStrategy,
  overwrite = FALSE,
  verbose = getFuncOption("verbose")
)
```

Arguments

...	Named arguments where each value is a function implementing an annotation method. The name becomes the method identifier (e.g., SingleR = SingleRAnnotate registers under key "SingleR"). Unnamed arguments are auto-named using their expression via get_names_4_ids().
registry	Target registry to store annotation methods. Default: SCAnnotateStrategy.
overwrite	Logical. If FALSE (default), throws an error when attempting to replace an existing method. Set to TRUE to allow updates.
verbose	Logical. Whether to print a success message upon registration. Default: inherits from package option getOption("SigBridgeRUtils.verbose").

Value

Invisibly returns TRUE on successful registration.

See Also

Other Registering: RegisterScreenMethod(), RegisterSeuratMethod()

Other Single_Cell_Annotation_Method: CellTypistAnnotate(), DownloadCellTypistModels(), ListCellTypistModels(), SCAnnotateStrategy, SingleRAnnotate(), mLLMCelltypeAnnotate()

Examples

```
## Not run:
# Register a custom annotation wrapper
MyAnnotator <- function(sc, ...) {
  # Custom logic returning annotated Seurat/SCE object
  sc
}

RegisterAnnoMethod(
  custom_annot = MyAnnotator,
  registry = SCAnnotateStrategy
)

# Attempting to re-register without `overwrite = TRUE` will fail
# RegisterAnnoMethod(custom_annot = AnotherAnnotator)  # Error!

## End(Not run)
```

RegisterScreenMethod *Register a Custom Screening Method for Phenotype-Driven Analysis*

Description

[Deprecated] Registers one or more user-defined screening functions into a shared registry (i.e., `ScreenStrategy`), enabling dynamic dispatch based on phenotype type (binary, survival, or continuous). Each method is stored with metadata including supported phenotypes and an optional parameter remapping function.

This function supports community-driven extension of analysis pipelines-ideal for integrating novel single cell phenotypic screening methods while maintaining compatibility with downstream validation and execution frameworks.

Usage

```
RegisterScreenMethod(
  ...,
  supported_phenotypes = c("binary", "survival", "continuous"),
  parameter_mapper = NULL,
  registry = ScreenStrategy,
  overwrite = FALSE,
  verbose = getFuncOption("verbose")
)
```

Arguments

`...`	Named arguments where each value is a function implementing a screening method. The name becomes the method identifier (e.g., `Scissor = DoScissor` registers `DoScissor` under the key `"Scissor"`). Unnamed arguments are auto-named using their expression (e.g., `Scissor` -> name `"Scissor"`).
`supported_phenotypes`	Character vector specifying which phenotype types the method supports. Must be a subset of `c("binary", "survival", "continuous")`. Default: all three.
`parameter_mapper`	A function that transforms the input parameter list before passing it to the executor. Useful for changing parameters from interface function. Receives a named list `params` and must return a modified list. Default: `NULL`.
`registry`	An environment used as a method registry (e.g., `ScreenStrategy`). New methods are stored as `registry[["MethodName"]]` = metadata list. Default: `ScreenStrategy`.
`overwrite`	Logical. If `FALSE` (default), throws an error when attempting to replace an existing method. Set to `TRUE` to allow updates.
`verbose`	Logical. Whether to print a success message upon registration. Default: inherits from package option.

Value

Invisibly returns `TRUE` on successful registration.

Registry Entry Structure

Each registered method is stored as a list with four elements:

- `method_name`: Character: the method key (e.g., "Scissor")
- `executor`: Function: the actual screening implementation
- `phenotypes`: Character vector: supported phenotype classes
- `mapper`: Function: parameter transformation hook

See Also

Other Add_Screen_method: `ScreenStrategy`, `TemplateScreenFunc()`, `ValidateScreenFunc()`

Other Registering: `RegisterAnnoMethod()`, `RegisterSeuratMethod()`

Examples

```
## Not run:
# Example: Register a custom screen method for survival data
DoMyScreen <- function(...) {
  # Implement your method here
  stop("Not implemented")
}

RegisterScreenMethod(
  # Key = function
  # jarl-ignore unreachable_code: example
  MyCox = DoMyScreen,
  # If `continuous` phenotype is not supported
  supported_phenotypes = c("survival", "binary"),
  parameter_mapper = function(params) {
    # If I prefer to use `quiet` instead of `verbose`
    params$quiet <- !params$verbose
    params
  }
)

## End(Not run)
```

RegisterSeuratMethod *Register a Seurat Processing Strategy*

Description

[Deprecated] Dynamically registers one or more new preprocessing strategies into a strategy environment (i.e., `SCPreProcessStrategy`). Each strategy is stored under a user-defined key and wraps a Seurat-compatible function to conform to the standard interface: `function(object, params)`.

Supports both direct function objects and character specifications of package-qualified function names (e.g., `h = "Seurat::RunHarmony"`). This enables safe, lazy resolution of external dependencies and simplifies pipeline configuration via strings.

Usage

```
RegisterSeuratMethod(
  ...,
  overwrite = FALSE,
  registry = SCPreProcessStrategy,
  verbose = getFuncOption("verbose")
)
```

Arguments

`...`	Named arguments where each name is a **strategy key** (e.g., `"h"`) and each value is either: • A **function**, or • A **character string** specifying a function (e.g., `"Seurat::RunHarmony"` or `"stats::prcomp"`). The function will be automatically wrapped to accept `object` and `params`, making it compatible with `SCPreProcess()` pipelines.
`overwrite`	Logical. If `FALSE` (default), throws an error when attempting to replace an existing strategy. Set to `TRUE` to allow updates.
`registry`	Environment. The target registry to store strategies. Default: `SCPreProcessStrategy`.
`verbose`	Logical. Whether to print registration messages. Default: inherits from `getOption("SigBridgeR.verbos`

Value

Invisibly returns `TRUE` on successful registration.

See Also

Other Registering: `RegisterAnnoMethod()`, `RegisterScreenMethod()`

Other single_cell_preprocess: `FilterTumorCell()`, `FindRobustElbow()`, `Pattern2Colname()`, `QCPatternDetect()`, `SCAnnotate()`, `SCIntegrate()`, `SCPreProcess()`, `SCPreProcessStrategy`

Examples

```
## Not run:
# Register using a package-qualified name
RegisterSeuratMethod(
  h = "Seurat::RunHarmony",
  registry = SCPreProcessStrategy
)

# Attempting to re-register without `overwrite = TRUE` will fail
# RegisterSeuratMethod(h = Seurat::RunHarmony)  # Error!

## End(Not run)
```

SCAnnotate *Unified Interface for Single-Cell Annotation Methods*

Description

A dispatch wrapper that routes cell type annotation requests to registered methods stored in SCAnnotateStrategy. Provides a consistent API for applying different annotation strategies (e.g., CellTypist, SingleR, mLLMCelltype) to a Seurat object without needing to call each method directly.

Usage

```
SCAnnotate(sc, ...)

## Default S3 method:
SCAnnotate(sc, ...)

## S3 method for class 'Seurat'
SCAnnotate(sc, method = c("CellTypist", "SingleR", "mLLMCelltype"), ...)
```

Arguments

sc	A Seurat object containing single-cell RNA-seq data with preprocessed expression matrix and (optionally) pre-computed clusters.
...	Method-specific parameters passed to the underlying annotation function. Examples: • For "SingleR": ref, labels, de.n, etc. • For "CellTypist": model, download, python, etc. • For "mLLMCelltype": tissue_name, models, api_keys, etc. See individual method documentation for details.
method	Character. Annotation method to use. Must be one of the keys registered in SCAnnotateStrategy (e.g., "CellTypist", "SingleR", "mLLMCelltype"). Partial matching is supported. Use names(SCAnnotateStrategy) to list available methods.

Value

The input Seurat object enriched with method-specific annotation metadata columns. Column names and content depend on the chosen method:

- SingleR: SingleR_labels, SingleR_delta_next, SingleR_pruned_labels
- CellTypist: celltypist_prediction, celltypist_probability, etc.
- mLLMCelltype: mllmcelltype_cell_type, mllmcelltype_consensus_proportion, mllmcelltype_entropy

Extensibility

New annotation methods can be registered via RegisterAnnoMethod or the unified Register interface. Once registered, they become immediately available through this dispatcher.

See Also

Other single_cell_preprocess: FilterTumorCell(), FindRobustElbow(), Pattern2Colname(), QCPatternDetect(), RegisterSeuratMethod(), SCIntegrate(), SCPreProcess(), SCPreProcessStrategy

Examples

```
## Not run:
# List available annotation methods
names(SCAnnotateStrategy)

# SingleR annotation with HPCA reference
annotated <- SCAnnotate(
  sc = seurat_obj,
  method = "SingleR",
  ref = "HPCA"
)

# CellTypist annotation with custom model
annotated <- SCAnnotate(
  sc = seurat_obj,
  method = "CellTypist",
  model = "Immune_All_Low",
  download = TRUE
)

# mLLMCelltype consensus annotation
annotated <- SCAnnotate(
  sc = seurat_obj,
  method = "mLLMCelltype",
  tissue_name = "Lung Tumor",
  models = c("gpt-4o", "claude-3-5-sonnet-20240620"),
  api_keys = list(
    openai = Sys.getenv("OPENAI_API_KEY"),
    anthropic = Sys.getenv("ANTHROPIC_API_KEY")
  )
)

# Partial matching works
annotated <- SCAnnotate(sc = seurat_obj, method = "single")  # -> "SingleR"

## End(Not run)
```

SCAnnotateStrategy *Registry of Cell Type Annotation Methods*

Description

An environment storing methods for annotating cell types.

Usage

```
SCAnnotateStrategy

AVAILABLE_ANNOTATE_METHODS
```

Details

Storing structure - named list

- `key`: method name
- `value`: list
  - `method_name`: method name
  - `executor`: function implementation of the method

See Also

Other Single_Cell_Annotation_Method: `CellTypistAnnotate()`, `DownloadCellTypistModels()`, `ListCellTypistModels()`, `RegisterAnnoMethod()`, `SingleRAnnotate()`, `mLLMCelltypeAnnotate()`

`SCPreProcess` *Single-Cell RNA-seq Preprocessing Pipeline*

Description

[Deprecated] A unified interface for standardized single-cell RNA-seq preprocessing. It accepts raw counts (matrix/data.frame), AnnData (via `anndata` or `anndataR`), or existing Seurat objects. The analysis flow is fully customizable via a character string `pipeline` and a configuration list `params`.

Usage

```
SCPreProcess(sc, ...)

## Default S3 method:
SCPreProcess(
  sc = NULL,
  ...,
  pipeline = "onsvpcetu",
 params = list(o = list(), n = list(), s = list(), v = list(), p = list(), e = list(), c
    = list(), t = list(), u = list()),
  quality_control = list(pattern = c("^MT-")),
 data_filter = list(nFeature_thresh = c(200L, 6000L), nCount_thresh = c(500L, 50000L),
    percent.mt = 20L, percent.rp = 60L),
  column2only_tumor = NULL
)
```

```
## S3 method for class 'R6'
SCPreProcess(
  sc,
  ...,
  pipeline = "onsvpcetu",
 params = list(o = list(), n = list(), s = list(), v = list(), p = list(), e = list(), c
    = list(), t = list(), u = list()),
  quality_control = list(pattern = c("^MT-")),
 data_filter = list(nFeature_thresh = c(200L, 6000L), nCount_thresh = c(500L, 50000L),
    percent.mt = 20L, percent.rp = 60L),
  column2only_tumor = NULL
)

## S3 method for class 'Seurat'
SCPreProcess(sc, column2only_tumor = NULL, ...)
```

Arguments

sc	Input data. Can be: • **Matrix/Data frame**: Raw count matrix (genes x cells). • **AnnData**: Python AnnData object (read via anndata or anndataR packages). • **Seurat**: A Seurat object (automatically validated and repaired if necessary).
...	Additional arguments for backward compatibility (mapped to params) or verbose control.
pipeline	A character string defining the processing steps and order. Characters map to Seurat functions: • 'o': CreateSeuratObject (Must be the first step and cannot be deleted) • 'n': NormalizeData • 's': ScaleData • 'v': FindVariableFeatures • 'p': RunPCA • 'e': FindNeighbors (Because "n" is used) • 'c': FindClusters • 't': RunTSNE • 'u': RunUMAP • 'r': SCTransform (Alternative to n/s/v) Default is "onsvpcetu".

params	A named list of lists containing arguments for each pipeline step. Keys match the pipeline characters (e.g., params$n for NormalizeData). Default structure:
	`list(`
	`  o = list(), # do not pass `counts``
	`  n = list(), # NormalizeData args`
	`  s = list(), # ScaleData args`
	`  v = list(), # FindVariableFeatures args`

```
  c = list(),
  ...
)
```

quality_control
: A list containing regex patterns for QC metric calculation. (See QCPatternDetect) Default: `list(pattern = "^MT-")`. Detected metrics (e.g., percent.mt) are added to meta.data.

data_filter
: A list of thresholds for cell filtering. Default: `assay` ("RNA"), `nFeature_RNA` (200-6000), `nCount_RNA` (500-50000), `percent.mt` (<20), `percent.rp` (<60). Only metrics detected via `quality_control` are filtered, i.e., nFeature_RNA, nCount_RNA and percent.mt.

column2only_tumor
: Optional character. Column name in metadata to filter for tumor cells (matches "Tumor", "Cancer", "Malignant", etc.). If `NULL`, no filtering is performed.

Details

Pipeline Strategy: The function parses the `pipeline` string and executes corresponding Seurat functions in order. To use `SCTransform`, simply change the pipeline string (e.g., `"orpetu"`) and provide parameters in `params$r`.

Quality Control & Filtering: QC metrics are generated based on regex patterns in `quality_control`. Cells are then filtered based on thresholds in `data_filter`. Column names for filtering are auto-generated (e.g., pattern "^MT-" -> filter "percent.mt"). If confused about the column name, use `SigBridgeR:::Pattern2Colname()`.

Value

A processed Seurat object with reductions, clusters, and QC metrics.

See Also

Other single_cell_preprocess: `FilterTumorCell()`, `FindRobustElbow()`, `Pattern2Colname()`, `QCPatternDetect()`, `RegisterSeuratMethod()`, `SCAnnotate()`, `SCIntegrate()`, `SCPreProcessStrategy`

Other input_preprocess: `DrawBulkPCA()`, `PhenoMap()`, `PhenoPreProcess()`, `SymbolConvert()`

Examples

```
## Not run:
# 1. Standard pipeline (LogNormalize -> Scale -> PCA -> UMAP)
obj <- SCPreProcess(
  sc = counts_matrix,
  pipeline = "onsvpcetu",
  params = list(c = list(resolution = 0.8))
)

# 2. SCTransform pipeline
obj_sct <- SCPreProcess(
  sc = counts_matrix,
  pipeline = "orpcu", # Create -> SCT -> PCA -> Clusters -> UMAP
```

```
  quality_control = list(pattern = c("^MT-", "^RP[LS]")),
  params = list(
    r = list(vars.to.regress = "percent.mt")
  )
)

# 3. Start from AnnData with tumor filtering
adata_object <- anndataR::read_h5ad("data.h5ad")
obj_ad <- SCPreProcess(
  sc = adata_object,
  column2only_tumor = "tissue"
)

## End(Not run)
```

`SCPreProcessStrategy` *Preprocessing Strategy Registry for Single-Cell Workflows*

Description

An environment that maps single-letter codes to standardized Seurat preprocessing operations. Each entry is a function of the form `function(...)` that wraps a core Seurat step (e.g., normalization, PCA, clustering)

This registry enables compact, composable pipeline definitions (e.g., via character strings like `"onvps"` for "Create -> Normalize -> VariableFeatures -> Scale -> PCA") and supports both standard (`NormalizeData`) and SCTransform-based workflows.

Usage

```
SCPreProcessStrategy
```

Available Operations

- o: Create Seurat object from count matrix (`SeuratObject::CreateSeuratObject`)
- n: Normalize data (`Seurat::NormalizeData`)
- v: Find variable features (`Seurat::FindVariableFeatures`)
- s: Scale data (`Seurat::ScaleData`)
- p: Run PCA (`Seurat::RunPCA`)
- c: Find clusters (`Seurat::FindClusters`)
- e: Find neighbors (`Seurat::FindNeighbors`)
- t: Run t-SNE (`Seurat::RunTSNE`)
- u: Run UMAP (`Seurat::RunUMAP`)
- r: SCTransform (`Seurat::SCTransform`)
- i: Integrate layers (`Seurat::IntegrateLayers`)

Usage Notes

- These functions are **not intended for direct interactive use**. They are internal building blocks for workflow engines (i.e., `SCPreProcess`).
- You can access any operation via `SCPreProcessStrategy$letter`, but doing so bypasses pipeline validation and error handling.
- To add more operations, use `RegisterSeuratMethod()`

See Also

Other single_cell_preprocess: `FilterTumorCell()`, `FindRobustElbow()`, `Pattern2Colname()`, `QCPatternDetect()`, `RegisterSeuratMethod()`, `SCAnnotate()`, `SCIntegrate()`, `SCPreProcess()`

`Screen` *Single-Cell Data Screening*

Description

Integrates matched bulk expression data and phenotype information to identify phenotype-associated cell populations in single-cell RNA-seq data using a registered screening method. Ensures consistency between bulk and phenotype data before analysis.

Usage

```
Screen(
  matched_bulk,
  sc_data,
  phenotype,
  label_type = NULL,
  phenotype_class = c("binary", "survival", "continuous"),
 screen_method = c("Scissor", "scPP", "scPAS", "scAB", "DEGAS", "LP_SGL", "PIPET",
    "SIDISH", "SCIPAC", "TiRank", "DEGASv2", "PhenoMapR"),
  ...
)
```

Arguments

`matched_bulk`	Matrix or data frame of preprocessed bulk RNA-seq expression data (genes x samples). Column names must match names/IDs in phenotype.
`sc_data`	A Seurat object containing scRNA-seq data to be screened.
`phenotype`	Phenotype data, either: • Named vector, where names match columns of matched_bulk • Patient survival data frame, where row names match columns of matched_bulk, and column names are "time" and "status"
`label_type`	Character specifying phenotype label type, e.g. "SBS1" or "time".

`phenotype_class`

Type of phenotypic outcome. Must be consistent with input data:

- "binary": Binary traits, e.g. case/control
- "continuous": Continuous measurements
- "survival": Survival information

`screen_method` Screening algorithm to use. Available options:

- "DEGASv2": see also `DoDEGASv2()`
- "Scissor": see also `DoScissor()`
- "TiRank": see also `DoTiRank()`
- "SCIPAC": see also `DoSCIPAC()`
- "scPP": see also `DoscPP()`
- "LP_SGL": see also `DoLP_SGL()`
- "PIPET": see also `DoPIPET()`
- "DEGAS": see also `DoDEGAS()`
- "scAB": see also `DoscAB()`
- "PhenoMapR": see also `DoPhenoMapR()`
- "SIDISH": see also `DoSIDISH()`
- "scPAS": see also `DoscPAS()`

`...` Additional method-specific parameters.

Scissor

- alpha: Numeric or NULL. Significance threshold. When NULL, alpha increases iteratively until corresponding cells are screened out. Default: 0.05.
- cutoff: Numeric. Threshold for terminating the iteration of alpha; only works when alpha = NULL. Default: 0.2.
- path2load_scissor_cache: Character. Default: NULL.
- path2save_scissor_inputs: Character. Path to save intermediary data. Can be loaded later via path2load_scissor_cache. Default: "Scissor_inputs.RData".
- reliability_test: Logical. Whether to perform reliability test. Default: FALSE.
- reliability_test.nfold: Integer. Cross-validation folds for reliability test. Default: 10.
- reliability_test.n: Integer. Number of cells to use for reliability test. Default: 10.
- cell_evaluation: Logical. Whether to perform cell evaluation. Default: FALSE.
- cell_evaluation.benchmark_data: .RData benchmark data for cell evaluation. Default: NULL.
- cell_evaluation.FDR: Numeric. FDR threshold for cell evaluation. Default: 0.05.
- cell_evaluation.bootstrap_n: Integer. Number of bootstrap samples for cell evaluation. Default: 10.

scPP

- ref_group: Integer or character. Reference group or baseline for binary comparisons, e.g. "Normal" or 0. Default: 0.

- Log2FC_cutoff: Numeric. Minimum log2 fold-change for binary markers. Default: 0.585.
- estimate_cutoff: Numeric. Effect size threshold for continuous traits. Default: 0.2.
- probs: Numeric. Quantile cutoff for cell classification. Default: 0.2.

scPAS

- assay: Character. Assay to use from sc_data. Default: "RNA".
- imputation: Logical. Whether to perform imputation. Default: FALSE.
- nfeature: Integer. Number of features to select. Default: 3000.
- alpha: Numeric or NULL. Significance threshold. When NULL, alpha increases iteratively until corresponding cells are screened out. Default: 0.01.
- independent: Logical. Whether the background distribution of risk scores is constructed independently of each cell. Default: TRUE.
- network_class: Character. Network class to use. Default: "SC" for single-cell-derived gene-gene similarity networks; another option is "bulk".
- permutation_times: Integer. Number of permutations. Default: 2000.
- FDR_threshold: Numeric. FDR threshold for identifying phenotype-associated cells. Default: 0.05.

scAB

- alpha: Numeric. Coefficient of phenotype regularization. Default: 0.005.
- alpha_2: Numeric. Coefficient of cell-cell similarity regularization. Default: 0.005.
- maxiter: Integer. NMF optimization iterations. Default: 2000.
- tred: Integer. Z-score threshold. Default: 2.

DEGAS

- sc_data.pheno_colname: Character. Phenotype column name in sc_data. Default: "NULL".
- select_fraction: Numeric. Fraction of cells to select for DEGAS. Default: 0.05.
- tmp_dir: Character. Temporary directory for DEGAS. Default: "NULL".
- env_params: List. Environment parameters for DEGAS. Default: list().
- degas_params: List. DEGAS parameters. Default: list().
- normality_test_method: Character. Normality test method for DEGAS. Default: "jarque-bera".

SIDISH

- sidish_params: List. SIDISH parameters. Default: list().
- env_params: List. Environment parameters for SIDISH. Default: list().

SCIPAC

- hvg: Integer. Number of highly variable genes to use for preprocessing. Default: 1000.
- do_pca_sc: Logical. If TRUE, first do PCA on sc.dat and use the rotation matrix on bulk.dat; if FALSE, first do PCA on bulk.data and use the rotation matrix on sc.dat. Default: FALSE.

- n_pc: Integer. Number of principal components to use. Default: 60.
- sc_batch_col: Character or vector. Batch variable for single-cell data. If character, should be a column name in `sc_data@metadata`. If vector, should match cell order. Default: NULL.
- resolution: Integer. Clustering resolution for cell type identification. Higher values produce more clusters. Default: 2.
- ela_net_alpha: Numeric. Elastic net mixing parameter (0 = ridge, 1 = lasso). Default: 0.4.
- bt_size: Integer. Bootstrap sample size for stability assessment. Default: 50.
- ncore: Integer. Number of CPU cores for parallel computation. Default: 7.
- ci_alpha: Numeric. Significance level for confidence intervals. Default: 0.05.
- nfold: Integer. Number of folds for cross-validation. Default: 10.
- assay: Character. Assay to use from sc_data. Default: "RNA".
- verbose: Logical. Whether to print progress messages. Default: inherits from getFuncOption("verbose").
- seed: Integer. Random seed for reproducibility. Default: 123.

PIPET

- group: Character or NULL. Metadata column used to stratify cells before screening. Default: NULL.
- discretize_method: Character. Strategy to binarize continuous phenotypes:
  - "median": Median threshold. Default.
  - "kmeans": Two-cluster k-means.
  - "custom": User-defined cutoffs via cutoff.
- cutoff: Numeric vector or NULL. Required when discretize_method = "custom". Specifies interior breakpoints on the normalized, log2-transformed phenotype scale, after scale(log2(x + 1)).
- label_type: Character. Phenotype label type stored in scRNA_data@misc. Default: "PIPET".
- log2FC: Numeric. Absolute log2 fold-change cutoff for bulk marker selection. Default: 1.
- p_adjust: Numeric. Adjusted p-value/FDR cutoff for marker selection. Default: 0.05.
- show_log2FC: Logical. Whether to annotate markers with signed log2FC direction. Default: TRUE.
- freq_counts: Integer or NULL. Minimum number of cells a gene must be expressed in. Default: NULL.
- normalize: Logical. Whether to apply LogNormalize before correlation. Default: TRUE.
- scale: Logical. Whether to center and scale gene expression before computing distances. Default: TRUE.
- nPerm: Integer. Number of label permutations. Default: 1000.
- distance: Character. Supported metrics: "cosine", "pearson", "spearman", "kendall", "euclidean", and "maximum". Default: "cosine".

- seed: Integer or NULL. Random seed. Default: inherits from getFuncOption("seed").
- verbose: Logical. Whether to print progress messages. Default: inherits from getFuncOption("verbose").
- parallel: Logical. Whether to enable parallel permutations. Requires future::plan() to be pre-set. Default: FALSE.

LP_SGL

- resolution: Numeric. Resolution parameter for Leiden clustering. Default: 0.6.
- alpha: Numeric. SGL parameter balancing L1 and L2 penalties. Default: 0.5.
- nfold: Integer. Number of folds for cross-validation. Default: 5.
- dge_analysis: List. Differential expression analysis settings:
  - run: Logical. Whether to run DEG analysis. Default: FALSE.
  - logFC_threshold: Numeric. Log fold-change threshold. Default: 1.
  - pval_threshold: Numeric. P-value threshold. Default: 0.05.

DEGASv2

- top_fraction_pos: Numeric. Proportion of cells to be labelled as "Positive". Default: 0.2.
- sclab: Character or NULL. Metadata column providing supervised cell labels. Default: NULL.
- bulk_hvg: Logical. Whether to select highly variable genes from bulk data. Default: TRUE.
- bulk_de: Logical. Whether to perform differential expression analysis on bulk data. Default: TRUE.
- sc_de: Logical. Whether to perform differential expression analysis on single-cell data. Default: TRUE.
- add_genes: Character vector or NULL. Additional genes to include in the analysis. Default: NULL.
- n_hvg: Integer. Number of highly variable genes to select. Default: 250.
- n_bulk_de: Integer. Number of bulk DE genes to select. Default: 250.
- n_sc_de: Integer. Number of single-cell DE genes to select. Default: 200.
- padj.thresh: Numeric. Adjusted p-value threshold for DE gene selection. Default: 0.05.
- only.pos: Logical. Whether to keep only positively enriched markers. Default: FALSE.
- min.pct: Numeric. Minimum fraction of cells expressing a gene for DE analysis. Default: 0.25.
- logfc.threshold: Numeric. Log fold-change threshold for DE analysis. Default: 0.25.
- n_st_classes: Integer. Number of single-cell classes. Default: length(unique(sc_data$seurat_clusters)
- loss_type: Character. Loss function for model training. One of "cross_entropy", "log_neg", "rank_loss". Default: "cross_entropy".

- transfer_type: Character. Domain adaptation method. One of "Wasserstein", "MMD". Default: "Wasserstein".
- model_save_dir: Character. Directory to save the trained model. Default: "DEGASv2_res".
- lambda1: Numeric. Weight of the classification loss. Default: 1.
- lambda2: Numeric. Weight of the domain-adaptation loss. Default: 3.
- lambda3: Numeric. Weight of the reconstruction loss. Default: 3.
- tot_seeds: Integer. Number of random seeds to train. Default: 10.
- tot_iters: Integer. Number of training iterations. Default: 300.
- extract_embs: Logical. Whether to extract cell embeddings. Default: FALSE.
- random_feat: Logical. Whether to use random features. Default: FALSE.
- random_perc: Numeric. Proportion of random features when random_feat = TRUE. Default: 0.8.
- early_stopping: Logical. Whether to enable early stopping. Default: FALSE.

TiRank

- tirank_params: List. TiRank algorithm parameters, including data preprocessing (validation_proportion, sampling_thresh, sampling_mode, top_var_genes, top_gene_pairs, p_value_threshold, max_cutoff, min_cutoff), neural network architecture (nhead, nhid1, nhid2, n_output, nlayers, n_pred, dropout, encoder_type, infer_mode), and training (n_trials, do_reject, tolerance). See `DoTiRank()` for details. Default: list().
- save_path: Character. Directory to save intermediate and final results. Soft-deprecated; prefer passing load_cache/save_cache via ... Default: "./TiRank_res".
- load_cache: Character or NULL. Path to cached TiRank data. Soft-deprecated; prefer passing load_cache via ... Default: NULL.
- verbose: Logical. Whether to print progress messages. Default: inherits from getFuncOption("verbose").
- seed: Integer. Random seed for reproducibility. Default: inherits from getFuncOption("seed").
- assay: Character. Assay to use from sc_data. Default: "RNA".
- save_cache: Character. Directory to save TiRank cache. Default: save_path.
- additional_description: Character. Optional description appended to the cache metadata.

PhenoMapR

- phenotype_column: Character or NULL. Outcome column name for binary/continuous phenotypes. Default: NULL (uses "phenotype_value").
- survival_time, survival_event: Character. Columns of the survival phenotype data frame. Default: "time" / "status".
- normalize: Logical. Whether to normalize bulk expression. Default: TRUE.
- platform: Character. "auto", "rnaseq", or "microarray". Default: "auto".
- probe_annotation: Probe-to-gene annotation for microarray data. Default: NULL.
- hugo_species: Character vector passed to HGNChelper. Default: c("human", "mouse").

- binary_positive_reference: Character. "second" or "first". Default: "second".
- assay, layer: Character. Seurat assay and layer used for scoring. Default: "RNA" / "data".
- z_score_cutoff: Numeric. Z-score cutoff for limiting paired genes. Default: 2.
- reference_sign: Integer. Sign applied to reference scores. Default: 1.
- score_mode: Character vector. Default: c("weighted_sum", "activity_adjusted").
- vars_to_regress: Character vector. Default: c("S.Score", "G2M.Score", "nCount_RNA").
- cancer_type: Character or NULL. Optional cancer-type context. Default: NULL.
- pseudobulk: Logical. Whether to score pseudobulk profiles. Default: FALSE.
- group_by: Character or NULL. Grouping variable for pseudobulk. Default: NULL.
- sample_metadata: Data frame or NULL. Metadata for pseudobulk. Default: NULL.
- percentile: Numeric. Tail percentile for phenotype groups. Default: 0.05.
- permutation_n: Integer or NULL. Number of permutations. Default: NULL.
- permutation_seed: Integer or NULL. Permutation seed; defaults to seed.

Value

A list containing:

- scRNA_data: A Seurat object with phenotype-associated cells labelled in a meta.data column.
- screen_result: Important information about the screened result related to the selected method.

Data Matching Requirements

- Column names of matched_bulk and names or row names of phenotype must be identical.
- Phenotype values must correspond to bulk samples, not directly to single cells.
- Mismatches will trigger an error before analysis begins.
- A built-in pre-run check is performed.

See Also

Associated functions:

- `DoScissor()`
- `DoscPP()`
- `DoscPAS()`
- `DoscAB()`
- `DoDEGAS()`
- `DoLP_SGL()`

- DoPIPET()
- DoSIDISH()
- DoSCIPAC()
- DoTiRank()
- DoDEGASv2()
- DoPhenoMapR()

ScreenFractionPlot *Visualization of Cell Screening Fractions*

Description

Generates stacked bar plots showing the proportion of cells classified as Positive/Negative/Neutral by single-cell screening algorithms (Scissor, scPAS, scPP, or scAB) across different sample groups. Supports both single and multiple screen types visualization.

Usage

```
ScreenFractionPlot(
  screened_seurat,
  group_by = "Source", # x-axis label
  screen_type = NULL, # default: search all available screens
  show_null = FALSE,
  plot_color = NULL, # stack bar color of each status
  show_plot = FALSE,
  plot_title = "Screen Fraction",
  stack_width = 0.85,
  x_text_angle = 45,
  axis_linewidth = 0.8,
  x_lab = NULL,
  y_lab = "Fraction of Status",
  ncol = 2, # number of columns for facet wrap
  nrow = NULL, # number of rows for facet wrap
  verbose = SigBridgeRUtils::getFuncOption("verbose"),
  ... # ggplot2 theme arguments
)
```

Arguments

screened_seurat	A Seurat object containing screening results in metadata. Must contain columns corresponding to screen_type.
group_by	Metadata column name for grouping samples (default: "Source").
screen_type	Screening algorithm(s) used. Can be a single value or vector. Must match metadata column(s) (case-sensitive, e.g., "scissor" for Scissor results).

show_null	Logical whether to show groups with zero cells (default: FALSE).
plot_color	Custom color palette (named vector format): • Required names: "Positive", "Negative", "Neutral" • Default: c("Neutral"="#CECECE", "Other"="#CECECE", Positive"="#c24b4b", "Negative"="#5189bb")
show_plot	Logical to immediately display plot (default: TRUE).
plot_title	Plot title (default: "Screen Fraction"). When multiple screen types, can be a vector of titles or single title (will append screen type prefix).
stack_width	Bar width (default: 0.85).
x_text_angle	X-axis label angle (default: 45).
axis_linewidth	Axis line thickness (default: 0.8).
x_lab	X-axis label (default: NULL).
y_lab	Y-axis label (default: "Fraction of Status").
ncol	Number of columns for facet wrap when multiple screen types (default: 2).
nrow	Number of rows for facet wrap when multiple screen types (default: NULL).
verbose	Logical, whether to print a message
...	Other arguments passed to ggplot2::theme()

Value

A list containing:

- stats: A data frame (single screen) or list of data frames (multiple screens) with screening statistics including:
  - Grouping variable counts
  - Raw cell counts
  - Percentage fractions
- plot: A ggplot2 object (single screen) or list of ggplot2 objects (multiple screens)
- combined_plot: A combined plot using patchwork (only for multiple screens)

Visualization Details

- Bars are ordered by descending Positive fraction
- Y-axis shows percentage (0-100%)
- Zero-fraction groups are automatically hidden unless show_null=TRUE
- For multiple screen types, plots can be combined using patchwork

See Also

Other visualization_function: ScreenUpset()

Examples

```
## Not run:
# Single screen type usage
res <- ScreenFractionPlot(
  screened_seurat = scissor_result,
  group_by = "PatientID",
  screen_type = "scissor"
)

# Multiple screen types at once
multi_res <- ScreenFractionPlot(
  screened_seurat = multi_screened_result,
  group_by = "TissueType",
  screen_type = c("scissor", "scPAS", "scPP", "scAB", "DEGAS"),
  ncol = 2
)

## End(Not run)
```

`ScreenMethod` *ScreenMethod: Screening Method Class for Phenotype-Associated Cell Detection*

Description

`ScreenMethod` is the S7 class for phenotype-associated cell screening methods. It extends Method with phenotype classification and mapper function support, and enforces a standardized function signature for all screening executors.

Usage

```
ScreenMethod(
  method_name,
  method_version,
  executor,
  phenotype_class,
  mapper = NULL,
  sigbridger_version = get_pkg_version()
)
```

Arguments

`method_name`	`character`. (Inherited from Method) A string identifying the screening method.
`method_version`	`character`. (Inherited from Method) A string indicating the version of the underlying software package.
`executor`	`function`. (Inherited from Method) The screening function. Must accept `matched_bulk`, `sc_data`, `phenotype`, `label_type`, and `phenotype_class` as arguments.

`phenotype_class`
: `character`. A character vector specifying the phenotype classification type(s) the method supports. Valid values: `"binary"`, `"continuous"`, `"survival"`. Default: all three.

`mapper`
: `function` or `NULL`. An optional function used to transform or map parameters before screening. When `NULL`, no mapper is applied.

`sigbridger_version`
: Package Version of SigBridgeR

Details

This class inherits from Method and adds screening-specific properties. It validates that the executor function accepts the required arguments: `matched_bulk`, `sc_data`, `phenotype`, `label_type`, and `phenotype_class`. This ensures a consistent interface across all screening algorithms (e.g., Scissor, scPAS, scPP, TiRank).

Value

A `ScreenMethod` S7 object.

See Also

Other SigBridgeR-Classes: `Method()`, `SigBridgeRBase()`

`ScreenMethodCache` *ScreenMethodCache: Screening Method Cache Container*

Description

`ScreenMethodCache` bundles cached screening data with its configuration and metadata. It serves as the primary container for reading from and writing to the cache system.

Usage

```
ScreenMethodCache(
  cache_path,
  cache_config_path,
  cache_data,
  screen_method_config,
  sigbridger_version = get_pkg_version()
)
```

Arguments

cache_path	character. Path to the cache directory where data files are stored.
cache_config_path	character. Path to the cache_config.json metadata file. Must end with "cache_config.json".
cache_data	list. A named list of cached data objects. Can be NULL or empty (for load-only use).
screen_method_config	A ScreenMethodConfig S7 object containing the screening method configuration.
sigbridger_version	Package Version of SigBridgeR

Details

This class is defined using the S7 object-oriented system and inherits from SigBridgeRBase. It wraps a ScreenMethodConfig together with cached data objects, file paths, and a read-only SigBridgeR version stamp.

Validator:

The class validator checks that:

- cache_path is an existing directory.
- The parent directory of cache_config_path exists.
- cache_config_path ends with "cache_config.json".

Value

A ScreenMethodCache S7 object.

Additional properties

@sigbridger_version character (read-only). The SigBridgeR package version at the time the cache was created. Automatically set from get_pkg_version() and cannot be modified by the user.

See Also

Other S7-Classes: AnnotationMethod(), BenchAccConfig(), BenchConfig(), BenchGSEAConfig(), BenchGeneScoreConfig(), BenchMemoryConfig(), ScreenMethodConfig(), ScreenMethodResult()

ScreenMethodConfig *ScreenMethodConfig: Screening Method Configuration*

Description

ScreenMethodConfig stores the configuration for a phenotype-associated cell screening method. It is used both to define screening parameters and as a cache result descriptor.

Usage

```
ScreenMethodConfig(
  method_name,
  param,
  method_version = NULL,
  phenotype_class = c("binary", "continuous", "survival"),
  label_type = character(0L),
  sigbridger_version = get_pkg_version()
)
```

Arguments

method_name	character. The name of the screening method (e.g., "Scissor", "scPAS"). Must match a key in ScreenStrategy.
param	list. A named list of parameters passed to the screening function. Must be non-empty.
method_version	character or NULL. The version of the underlying software package. When NULL, automatically resolved via r_pkg_version(method_name).
phenotype_class	character. The phenotype classification type(s). One or more of "binary", "continuous", "survival". Default: all three.
label_type	character. The label type for the screening method. Default: character(0L).
sigbridger_version	Package Version of SigBridgeR

Details

This class is defined using the S7 object-oriented system and inherits from SigBridgeRBase. It validates that method_name is a registered screening method in ScreenStrategy and that phenotype_class is one of the supported types.

Validator:

The class validator enforces:

- param must be a non-empty named list.
- phenotype_class must be one or more of "binary", "continuous", "survival".
- method_name must match a key in ScreenStrategy.

Value

A ScreenMethodConfig S7 object.

See Also

Other S7-Classes: AnnotationMethod(), BenchAccConfig(), BenchConfig(), BenchGSEAConfig(), BenchGeneScoreConfig(), BenchMemoryConfig(), ScreenMethodCache(), ScreenMethodResult()

ScreenMethodResult *ScreenMethodResult: Screening Method Result Container*

Description

ScreenMethodResult stores the results of a phenotype-associated cell screening analysis. It wraps a Seurat object containing the screened single-cell data and supports attaching additional dynamic properties (e.g., screening statistics, metadata) via ... in the constructor.

Usage

```
ScreenMethodResult(scRNA_data, ..., sigbridger_version = get_pkg_version())
```

Arguments

scRNA_data	A Seurat object containing the screening results. This is the core required property of the class.
...	Named arguments to set as additional S7 properties on the result object (e.g., screening statistics, method-specific outputs). Stored as a list in the extra property.
sigbridger_version	Package Version of SigBridgeR

Details

This class is defined using the S7 object-oriented system and inherits from SigBridgeRBase. The core required property is scRNA_data, which must be a Seurat object containing the screening results.

Additional properties can be attached dynamically via the constructor's ... argument. Named arguments are set as S7 properties on the returned object, allowing flexible result storage without subclassing.

Validator:

The class validator enforces:

- scRNA_data must be present on the object.
- scRNA_data must be a Seurat object.

Value

A ScreenMethodResult S7 object.

See Also

Other S7-Classes: AnnotationMethod(), BenchAccConfig(), BenchConfig(), BenchGSEAConfig(), BenchGeneScoreConfig(), BenchMemoryConfig(), ScreenMethodCache(), ScreenMethodConfig()

ScreenStrategy *Registry of Phenotype-Associated Cell Screening Methods*

Description

An environment storing methods for screening phenotype-associated cells.

Usage

```
ScreenStrategy

AVAILABLE_SCREEN_METHODS
```

Details

Storing structure - named list

- key: method name
- value: list
  - method_name: method name
  - executor: function implementation of the method
  - phenotype_class: The phenotype types supported by this method
  - mapper: A function that transforms the parameters passed to the Screen function before forwarding them to the executor; both input and output must be of type list.

See Also

Other Add_Screen_method: RegisterScreenMethod(), TemplateScreenFunc(), ValidateScreenFunc()

ScreenUpset *ScreenUpset - Visualize cell type intersections from screened Seurat object*

Description

This function creates an UpSet plot to visualize intersections between different screening methods (e.g., scissor, scAB, scPAS, scPP) in a Seurat object metadata. It calculates all possible combinations of screening types and displays the number of cells positive for each combination.

Usage

```
ScreenUpset(
  screened_seurat,
  screen_type = NULL,
  order_by = c("freq", "degree"),
  show_plot = FALSE,
  n_intersections = 20L,
  bar_color = "#5189bb",
  combmatrix_point_color = "black",
  verbose = SigBridgeRUtils::getFuncOption("verbose"),
  ...
)
```

Arguments

screened_seurat	A Seurat object containing screening results in metadata. Must contain columns with screening types marked as "Positive".
screen_type	Character vector of screening types to analyze. Default: NULL, indicating that a self-search pattern will be used.
order_by	Character vector of columns to order the intersections by. Default: c("freq", "degree").
show_plot	Whether to show the upset plot. Default: FALSE
n_intersections	Number of intersections to display in the plot. Default: 20.
bar_color	Color for the bars in the plot. Default: "#5189bb".
combmatrix_point_color	Color for points in the combination matrix. Default: "black".
verbose	Logical, whether to print a message
...	Additional arguments passed to `ggupset::scale_x_upset()`, `ggupset::theme_combmatrix()`, and `ggplot2::theme()` for customizing the plot appearance. Argumetns are auto-filtered

Details

The function performs the following steps:

1. Validates input parameters and checks if specified screening types exist in metadata
2. Generates all possible combinations of screening types (from 1 to the total number of types)
3. Creates a logical matrix of positive cells for efficient computation
4. Calculates cell counts for each intersection using vectorized operations
5. Creates an UpSet plot visualization with customizable appearance

Value

A list containing two elements:

- `plot`: A ggplot object displaying the UpSet plot
- `stats`: A data frame with intersection statistics including:
  - `intersection`: Name of the intersection
  - `sets`: List of screening types in the intersection
  - `count`: Number of cells positive for all screening types in the intersection

See Also

Other visualization_function: `ScreenFractionPlot()`

Examples

```
## Not run:
# Basic usage with default parameters
result <- ScreenUpset(screened_seurat = my_seurat_obj)

# Customize screening types and appearance
result <- ScreenUpset(
  screened_seurat = my_seurat_obj,
  screen_type = c("scissor", "scAB"),
  n_intersections = 15,
  title = "Custom Title",
  bar_color = "darkred"
)

## End(Not run)
```

`setFuncOption` *Configuration Functions for SigBridgeR Package*

Description

These functions provide a centralized configuration system for the SigBridgeR package, allowing users to set and retrieve package-specific options with automatic naming conventions.

setFunOption sets one or more configuration options for the SigBridgeR package. Options are automatically prefixed with "SigBridgeR." if not already present, ensuring proper namespace isolation.

getFuncOption retrieves configuration options for the SigBridgeR package. The function automatically handles the "SigBridgeR." prefix, allowing users to reference options with or without the explicit prefix.

checkFuncOption validates configuration options for the SigBridgeR package. This function ensures that all options meet type and value requirements before they are set in the global options.

Usage

```
setFuncOption(...)
```

Arguments

...	Named arguments representing option-value pairs. Options can be provided with or without the "SigBridgeR." prefix. If the prefix is missing, it will be automatically added.

Details

This function performs the following validations:

verbose Must be single logical values (TRUE/FALSE)

timeout, seed Must be single integer values

nthreads Must be a single positive integer value

The function is automatically called by setFuncOption to ensure all configuration options are valid before they are set.

Value

Invisibly returns NULL. The function is called for its side effects of setting package options.

The value of the specified option if it exists, otherwise the provided default value.

No return value. The function throws an error if the value doesn't meet the required specifications for the given option.

Available Opions

- verbose: A logical value indicating whether to print verbose messages, defaults to TRUE
- timeout: An integer specifying the timeout in seconds for parallel processing, defaults to ‘180L“
- seed: An integer specifying the random seed for reproducible results, defaults to ‘123L“
- nthreads: An integer specifying the number of threads for parallel processing, defaults to ‘1L“

Examples

```
# Set options with automatic prefixing
setFuncOption(verbose = TRUE)

# Retrieve options with automatic prefixing
getFuncOption("verbose")
```

SetThreads *Configure Parallel Execution Backends*

Description

Unified interface to configure thread counts and acceleration features across system-level (OpenMP/data.table/cheapr) and TensorFlow backends. Uses a hierarchical configuration model where global threads serves as default for all backends unless explicitly overridden.

Critical: TensorFlow configuration must be set *before* importing TensorFlow via reticulate::import("tensorflow").

Usage

```
SetThreads(
  threads = getFuncOption("nthreads"),
  dt = threads,
  cheapr = threads,
  qs2 = threads,
  wgcna = max(threads, 2L),
  seurat = threads,
  openmp = NULL,
 tf_config = list(xla_flag = "--tf_xla_auto_jit=2 --tf_xla_cpu_global_jit", xla_device =
    NULL, inter_op = NULL, intra_op = c(1L, NULL)),
  verbose = getFuncOption("verbose"),
  ...
)
```

Arguments

threads	Integer. Global thread count used as default for all backends. Default: 2L
dt	Integer. Thread count for data.table (default: inherited from threads).
cheapr	Integer. Thread count for cheapr (default: inherited from threads).
qs2	Integer. Thread count for qs2 (default: inherited from threads).
wgcna	Integer. Thread count for WGCNA (default: inherited from max(threads, 2L)). Values below 2 are raised to 2, because WGCNA::enableWGCNAThreads() requires at least two threads.
seurat	Integer. Thread count for Seurat (default: inherited from threads). Only applied when **Seurat** >= 6.0.0 is installed, as Seurat::setThreads() is not available in earlier versions.
openmp	Integer. Thread count for OpenMP (default: inherited from threads).
tf_config	Named list for TensorFlow-specific configuration: • xla_flag: Character. XLA JIT compilation flags (default: auto-optimized) • xla_device: Integer. XLA device ID (default: 1L) • inter_op: Integer. Inter-op parallelism threads (default: max(2, floor(threads/4))) • intra_op: Integer. Intra-op parallelism threads (default: 1L). If NULL, inherits global threads by default.
verbose	Logical. Whether to print verbose output (default: inherited from function options).
...	Additional arguments for data.table::setDTthreads() (e.g., restore).

Value

Invisible list with old/new values per backend.

Examples

```
## Not run:
# Basic usage: auto-detect and configure
SetThreads()

# Explicit thread count for CPU-intensive workloads
SetThreads(threads = 12L)

# TensorFlow-optimized configuration for deep learning
SetThreads(
  threads = 8L,
  tf_config = list(
    inter_op = 2L,
    intra_op = 8L
  )
)
# Import AFTER SetThreads()

# Configure only data.table for memory-efficient workflows
```

```
SetThreads(dt = 4L , verbose = FALSE)

## End(Not run)
```

SetupPyEnv *Create or Use Python Environment with Required Packages*

Description

Sets up a Python environment with specified packages. This function can create new environments or reuse existing ones, supporting both Conda and venv environment types. It ensures all required dependencies are properly installed and verified.

Default method for unsupported environment types. Throws an informative error with supported environment types.

Usage

```
SetupPyEnv(env_type = c("conda", "venv"), ...)
```

Arguments

env_type	Character string specifying the type of Python environment to create or use. One of: "conda", "venv".
...	Additional parameters passed to specific environment methods.

Details

This function provides a comprehensive solution for Python environment management in R projects, particularly for machine learning workflows requiring TensorFlow. Key features include:

- **Environment Creation**: Automatically creates new environments or reuses existing ones with the same name
- **Package Management**: Installs specified Python packages with version pinning support
- **Verification**: Validates environment setup and package installations
- **Flexible Methods**: Supports different backend methods for environment creation (reticulate vs system calls)

The function uses S3 method dispatch to handle different environment types, allowing for extensible support of additional environment managers in the future.

Value

A data frame containing verification results for the environment setup, including installation status of all required packages. Invisibly returns the verification results.

See Also

reticulate::conda_create(), reticulate::virtualenv_create() for underlying environment creation functions.

Examples

```
## Not run:
# Setup a Conda environment with default parameters
SetupPyEnv("conda")

# Setup a venv environment
SetupPyEnv("venv")

## End(Not run)
```

SigBridgeR-properties *SigBridgeR Base Class Properties*

Description

These are S7 property definitions used within the SigBridgeRBase class and its subclasses. They define the types, validators, and default values for class attributes. These properties are exported for community developers to extend and subclass SigBridgeRBase.

Usage

```
property_sigbridger_verison

property_verison

property_chr

property_phenotype_class

property_fn

property_list

property_mapper_fn

property_py_env

property_data_list

property_seurat

property_phenotype

property_matrix
```

Format

An S7 property of class `class_character`.

An S7 property of class `class_character`.

An S7 property of class `class_character`.

An S7 property of class `class_character`.

An S7 property of class `class_function`.

An S7 property of class `class_list`.

An S7 property of class `class_any`.

An S7 property of class `class_any`.

An S7 property of class `class_list`.

An S7 property of class `class_any`.

An S7 property of class `class_any`.

An S7 property of class `class_any`.

Details

A read-only character property that returns the current version of the `SigBridgeR` package. Attempting to set this property will raise an error.

The version string is validated to contain at least one dot (`.`). The default value is obtained from `get_pkg_version()`.

A read-only character property that returns the current version of the `SigBridgeR` package. Attempting to set this property will raise an error.

The version string is validated to contain at least one dot (`.`). The default value is obtained from `get_pkg_version()`.

A general-purpose character property with no additional validation. Can be used for any string-typed attribute in subclasses.

A character property that specifies the phenotype classification type(s). Valid values are `"binary"`, `"continuous"`, and `"survival"`. Multiple values can be provided as a character vector.

Default: `c("binary", "continuous", "survival")`.

A property that accepts a function value. Used to store callback or transformation functions in class attributes.

A named list property. The value must be a named list (not `NULL` or `NA`) with unique names. Duplicate names are not allowed.

A property that accepts either a function or `NULL`. Used for storing mapper/transformation functions that may be optionally specified.

If a value is provided, it must be a function; otherwise, `NULL` is accepted to indicate no mapper is configured.

A property representing a Python environment configuration. Accepts either a character string (e.g., a conda environment name or path) or `NULL` (when no Python environment is specified).

A named list property for storing multiple data objects. The list must be named (if not `NULL`) and names must be unique.

Unlike property_list, this property allows NULL, NA, and empty lists (list()) as valid values, making it suitable for optional data storage.

A property that requires a Seurat object. The value must inherit from the "Seurat" class.

This property is used to store single-cell data in Seurat format for downstream analysis.

A property that accepts a phenotype data

A property that accepts a matrix.

See Also

SeuratObject::Seurat

SigBridgeR-S3 *Base R Methods for SigBridgeR Objects*

Description

S3 methods that make every SigBridgeRBase object – and therefore every SigBridgeR object, such as ScreenMethod, ScreenMethodResult, and InterceptStrategy – behave like a standard R object. They support the base generics print(), length(), names(), names<-(), format(), as.list(), and as.character().

Pretty-print a SigBridgeR object using the cli package.

The class name is displayed in bold blue. Slots are displayed as an unordered list, with slot names in regular blue and each value preceded by its type.

Returns the number of properties (slots) stored in a SigBridgeRBase object.

Returns the names of all properties (slots) stored in a SigBridgeRBase object.

Returns the names of all properties (slots) stored in a SigBridgeRBase object.

Format a SigBridgeRBase object as a compact character vector.

Converts a SigBridgeRBase object into a named list of its properties.

Coercion of SigBridgeRBase to character is intentionally unsupported. Use format() to obtain a character representation instead.

Usage

```
## S3 method for class '`SigBridgeR::SigBridgeRBase`'
print(x, ...)

## S3 method for class '`SigBridgeR::SigBridgeRBase`'
summary(object, ...)

## S3 method for class '`SigBridgeR::SigBridgeRBase`'
length(x)

## S3 method for class '`SigBridgeR::SigBridgeRBase`'
```

```
names(x)

## S3 replacement method for class '`SigBridgeR::SigBridgeRBase`'
names(x) <- value

## S3 method for class '`SigBridgeR::SigBridgeRBase`'
format(x, ...)

## S3 method for class '`SigBridgeR::SigBridgeRBase`'
as.list(x, ...)

## S3 method for class '`SigBridgeR::SigBridgeRBase`'
as.character(x, ...)
```

Arguments

x	A SigBridgeRBase object. Basically all SigBridgeR objects.
...	Additional arguments passed to the method. Currently unused.
object	A SigBridgeRBase object. Basically all SigBridgeR objects.
value	A character vector of property names.

Details

All SigBridgeR objects inherit from the abstract base class SigBridgeRBase and store their data in S7 properties. The methods documented on this page expose those properties through familiar base R interfaces:

- `print()` displays a human-readable summary of the object. The class name is shown in bold blue and each property is listed with its name, value type, and value.
- `length()` returns the number of properties stored in the object.
- `names()` returns the names of the object's properties.
- `names<-()` is restricted: properties are validated by their S7 definitions, so modifying them in place is not allowed. Reconstruct a new object instead.
- `format()` converts the object to a compact character vector, starting with a header of the form `<Class: SigBridgeR <version>>`.
- `as.list()` converts the object to a named list of its properties.

`as.character()` is intentionally unsupported: coercing a SigBridgeRBase object to a character vector is not meaningful and always throws an error. Use `format()` instead.

The class name is displayed in bold blue. Slots are displayed as an unordered list, with slot names in regular blue and each value preceded by its type.

The first line is displayed as `<Class: name version>` when the object has `method_name` and `method_version` properties, for example `<ScreenMethod: SigBridgeR 4.0.0>`. Each following line contains one property name and its value on the same line. Property types are not shown.

Value

Invisibly returns `x`.

Invisibly returns `object`.

An integer: the number of properties.

A character vector of property names.

A character vector of property names.

A character vector.

A named list of properties.

Throws an error; never returns.

See Also

- SigBridgeRBase for the abstract base class on which these methods dispatch.
- `S7::props()` for the underlying property accessor.

`SigBridgeRBase` *SigBridgeRBase: Abstract Base Class for SigBridgeR*

Description

`SigBridgeRBase` is the abstract base class for all S7 classes in the SigBridgeR package. It provides a unified foundation for S3 method dispatch and ensures consistent behavior across all subclasses.

Usage

```
SigBridgeRBase(sigbridger_version = "4.0.0")
```

Arguments

`sigbridger_version`
: Package Version of SigBridgeR

Details

This class is defined using the S7 object-oriented system and is marked as abstract, meaning it cannot be instantiated directly. All other classes in the SigBridgeR package should inherit from `SigBridgeRBase`.

The primary purpose of this base class is to serve as a dispatch target for base R S3 generic functions, enabling polymorphic behavior across all SigBridgeR S7 objects through S3 method dispatch.

See Also

Other SigBridgeR-Classes: `Method()`, `ScreenMethod()`

SingleRAnnotate *Annotate Single-Cell Data Using SingleR*

Description

Performs automated cell type annotation using the SingleR package. Supports built-in references (e.g., Human Primary Cell Atlas) or user-provided SingleCellExperiment objects. Integrates seamlessly with Seurat objects by adding prediction results as metadata columns.

Usage

```
SingleRAnnotate(sc, ...)

## Default S3 method:
SingleRAnnotate(sc, ...)

## S3 method for class '`SigBridgeR::ScreenMethodResult`'
SingleRAnnotate(sc, ...)

## S3 method for class 'Seurat'
SingleRAnnotate(sc, ...)

## S3 method for class 'SingleCellExperiment'
SingleRAnnotate(
  sc,
  verbose = getFuncOption("verbose"),
  ensembl = FALSE,
  cell.ont = c("all", "nonna", "none"),
  legacy = FALSE,
  ref = c("HPCA", "custom"),
  labels = NULL,
  ...
)
```

Arguments

sc	A single-cell dataset, either: • A Seurat object (recommended), or • A SingleCellExperiment object, or • A ScreenMethodResult object, whose scRNA_data slot is annotated.
...	Additional arguments passed to SingleR::SingleR() (e.g., assay.type.ref, genes).
verbose	Logical. Whether to print progress messages. Default: inherits from package option getOption("SigBridgeR.verbose").
ensembl	Logical scalar indicating whether to convert row names to Ensembl IDs. Genes without a mapping to a non-duplicated Ensembl ID are discarded.

`cell.ont` String specifying whether Cell Ontology terms should be included in the colData. If `"nonna"`, all samples without a valid term are discarded; if `"all"`, all samples are returned with (possibly NA) terms; if `"none"`, terms are not added.

`legacy` Logical scalar indicating whether to pull data from ExperimentHub. By default, we use data from the gypsum backend.

`ref` Reference data for annotation. One of:

- `"HPCA"`: Uses the *Human Primary Cell Atlas* from the `celldex` package.
- A `SingleCellExperiment` object containing pre-labeled reference cells.

Note: The placeholder option `"custom"` is not valid-users must provide an actual reference object.

A numeric matrix of (usually normalized and log-transformed) expression values from a reference dataset, or a SummarizedExperiment object containing such a matrix; see `SingleR::trainSingleR` for details. Alternatively, a list or List of SummarizedExperiment objects or numeric matrices containing multiple references. Row names may be different across entries but only the intersection will be used,

`labels` A character vector of cell type labels corresponding to the columns of `ref`. Required when `ref` is a custom `SingleCellExperiment`. For `"HPCA"`, defaults to `ref$label.main`.

Alternatively, if ref is a list, labels should be a list of the same length. Each element should contain a character vector or factor specifying the labels for the columns of the corresponding element of ref.

Value

- If input is a `Seurat` object: returns the same object with three new metadata columns:
  - `SingleR_labels`: Predicted cell type labels.
  - `SingleR_delta_next`: Confidence score (difference between top and second-best scores).
  - `SingleR_pruned_labels`: Labels after pruning low-confidence assignments.
- If input is a `SingleCellExperiment`: returns the `SingleR` prediction result
- If input is a ScreenMethodResult: returns the annotated `Seurat` object from its `scRNA_data` slot, as above.

Requirements

- The `celldex` package is required when using `ref = "HPCA"`.
- All gene names must be consistent between test and reference datasets (e.g., Ensembl or symbol).

See Also

Other Single_Cell_Annotation_Method: `CellTypistAnnotate()`, `DownloadCellTypistModels()`, `ListCellTypistModels()`, `RegisterAnnoMethod()`, `SCAnnotateStrategy`, `mLLMCelltypeAnnotate()`

Examples

```
## Not run:
# Using HPCA reference (requires celldex)
annotated_seurat <- SingleRAnnotate(seurat_obj, ref = "HPCA")

# Using custom reference
library(celldex)
ref_sce <- BlueprintEncodeData()
annotated_sce <- SingleRAnnotate(
  sc = my_sce,
  ref = ref_sce,
  labels = ref_sce$label.main,
  de.n = 30
)

## End(Not run)
```

`SymbolConvert` *Convert Ensembles Version IDs & TCGA Version IDs to Genes in Bulk Expression Data*

Description

Convert Ensembles version IDs and TCGA version IDs to genes. NA values (unmatched IDs) are replaced with the original unconverted IDs (Ensembl version IDs or TCGA version IDs).

Usage

```
SymbolConvert(
  data,
  genome_build = c("hg38", "hg19", "mm10", "mm9", "ce11", "T2T"),
  update_symbol = TRUE,
  update_symbol_method = c("auto", "scCustomize", "Seurat"),
  verbose = SigBridgeRUtils::getFuncOption("verbose"),
  unknown_format = lifecycle::deprecated(),
  ...
)
```

Arguments

`data`	RNA expression data with rownames as Ensembles version IDs or TCGA version IDs(matrix or data.frame)
`genome_build`	Genome build of the data. Default: `"hg38"`.
`update_symbol`	Whether to update gene symbol to latest version. Default: `TRUE`.
`update_symbol_method`	Method used to update gene symbols, one of `"auto"`, `"scCustomize"` or `"Seurat"`. `"auto"` prefers `scCustomize` when it is installed, otherwise falls back to `Seurat`. Only used when `update_symbol = TRUE`. Default: `"auto"`.

verbose	Whether to print verbose messages. Default: TRUE.
unknown_format	**[Deprecated]** This argument is deprecated and ignored. NA values are now replaced with the original unconverted IDs.
...	No usage

Value

Data with rownames converted to gene symbols

See Also

Other input_preprocess: DrawBulkPCA(), PhenoMap(), PhenoPreProcess(), SCPreProcess()

TemplateScreenFunc *Generate a Template Screening Function with Optional Roxygen2 Documentation*

Description

Creates a scaffold for a new screening function that conforms to the expected interface for bulk-scRNA integration workflows. The template includes standard parameters (e.g., matched_bulk, sc_data, phenotype) and a return structure compatible with downstream validation (e.g., ValidateScreenFunc). Optionally includes roxygen2 documentation and supports writing to a new or existing .R file.

When filename = "current", the function attempts to append to the active script in RStudio or Positron (requires rstudioapi).

Usage

```
TemplateScreenFunc(
  filename = NULL,
  func_name = "my_screen",
  append = TRUE,
  documentation = FALSE,
  open = rlang::is_interactive(),
  ...
)
```

Arguments

filename	Character. Path to the output .R file. If "current" (case-insensitive), uses the currently open script in RStudio/Positron. Must end in .R if specified explicitly. Default: "my_screen.R".
func_name	Character. Name of the generated function. Must be a valid R identifier (starts with letter, followed by letters, digits, _, or .). Default: "my_screen".
append	Logical. If TRUE (default), appends to an existing file; otherwise, overwrites it. If overwriting a non-empty file, prompts for user confirmation.

documentation	Logical. Whether to include a full roxygen2 documentation block above the function. Default: FALSE. If TRUE, users should run devtools::document() afterward.
open	Logical. Whether to open the file in the RStudio/Positron editor after writing (if available). Default: TRUE.
...	Additional arguments (currently unused, reserved for future extension).

Value

Invisibly returns the path to the written file (filename).

See Also

Other Add_Screen_method: RegisterScreenMethod(), ScreenStrategy, ValidateScreenFunc()

Examples

```
## Not run:
# Create a new screen function with documentation
TemplateScreenFunc(
  filename = "custom_screens.R",
  func_name = "MyCoxScreen",
  documentation = TRUE
)

# Append to current script in RStudio
TemplateScreenFunc(filename = "current", func_name = "QuickFilter")

## End(Not run)
```

ValidateScreenFunc *Validate Custom Screening Function Compliance*

Description

Verifies if a user-provided function meets the interface requirements for the screening pipeline.

Usage

```
ValidateScreenFunc(func, ...)
```

Arguments

func	Function to validate. Must accept core parameters and return expected structure.
...	For future updates

Value

TRUE if all checks pass, otherwise terminates with diagnostic report

See Also

Other Add_Screen_method: RegisterScreenMethod(), ScreenStrategy, TemplateScreenFunc()

Examples

```
## Not run:
# Example compliant function
my_screen <- function(
  sc_data,
  matched_bulk,
  phenotype,
  label_type,
  phenotype_class,
  verbose = FALSE,
  ...
) {
  if (verbose) {
    message("Running custom screen")
  }
  list(
    scRNA_data = sc_data, # Must be Seurat object
    results = data.frame(score = runif(10))
  )
}

ValidateScreenFunc(my_screen)

## End(Not run)
```

WeightedVote *Weighted Voting Aggregation for Multi-Voter Classification*

Description

Aggregates classification votes from multiple voters using weighted scoring. Each voter assigns one of four labels ("Positive", "Negative", "Neutral", "Other") to each item, and votes are combined using user-specified weights.

Usage

```
WeightedVote(
  vote_data = data.frame(),
  weights = double(),
  ties.method = c("random", "first", "last")
)
```

Arguments

vote_data	A data frame or matrix where: • Rows represent items to be classified • Columns represent individual voters • Cell values must be one of: "Positive", "Negative", "Neutral", "Other" Row names (if present) are preserved in the output.
weights	Named numeric vector specifying the weight for each voter. Names must exactly match colnames(vote_data). Weights should be non-negative, with higher values indicating greater influence.
ties.method	Character string specifying how to break ties. Must be one of "random", "first", or "last".

Value

Character vector of final aggregated labels, one per row in vote_data. Each element is one of "Positive" or "Other" (the winner with highest weighted score). If input has row names, they are preserved as names in the output vector.

Examples

```
# Example 1: Simple majority voting
vote_data <- data.frame(
  Expert1 = c("Positive", "Negative", "Neutral", "Positive"),
  Expert2 = c("Positive", "Neutral", "Other", "Negative"),
  Expert3 = c("Negative", "Positive", "Neutral", "Positive"),
  row.names = c("Cell1", "Cell2", "Cell3", "Cell4")
)

weights <- c(Expert1 = 1, Expert2 = 1, Expert3 = 1)
result <- WeightedVote(vote_data, weights)
```

WriteCache *Write an R Object to Cache*

Description

Writes an R object to the cache directory. Uses qs2 for fast serialization when available, falling back to rdata for general objects. Small data frames and data.tables are saved as CSV via data.table::fwrite() for easy inspection.

Usage

```
WriteCache(cache, ...)
```

Arguments

cache	A ScreenMethodConfig or ScreenMethodCache S7 object containing the data and configuration to write.
...	Additional arguments passed to methods. For ScreenMethodConfig: • file: character(1). Path to the output cache_config.json file. • additional_description: character or NULL. An optional string describing the cache entry. Default: NULL. • verbose: logical. Whether to print progress messages. Default: TRUE. For ScreenMethodCache: • format: character. Cache format. One of "auto", "qs2", "qdata", "rdata", or "csv". When "auto" default, the format is detected per object. • max_rows: integer. Maximum number of rows to consider a data.frame small for CSV output. Default: 1000L. • max_cols: integer. Maximum number of columns to consider a data.frame small for CSV output. Default: 20L. • additional_description: character or NULL. An optional string describing the cache entry. Default: NULL. • verbose: logical. Whether to print progress messages. Default: TRUE.

Details

The ScreenMethodCache method writes both the cache metadata (via WriteCacheMeta()) and the cached data objects to disk. The ScreenMethodConfig method only writes the metadata.

Format detection (ScreenMethodCache):

When format = "auto" (default), the format for each data object is detected automatically:

- If the object is a 2D structure (data.frame/matrix) with at most max_rows rows and max_cols columns, CSV format is used.
- If qs2 is installed and the object is a list, vector, data.frame, or matrix, qdata format is used.
- If qs2 is installed but the object is not qdata-compatible, qs2 format is used.
- Otherwise, rdata format is used as a fallback.

Value

Invisible. Returns the absolute path(s) to the written cache file(s).

Methods

WriteCache is an S7 generic with methods available for the following classes:

- ANY
- SigBridgeR::ScreenMethodCache
- SigBridgeR::ScreenMethodConfig

See Also

Other cache_config: CacheSetHere(), CacheSysCall(), CheckCache(), ChooseCache(), LoadCache(), WriteCacheMeta()

Examples

```
## Not run:
# Write using a ScreenMethodCache
cache <- ScreenMethodCache(
  cache_path = "Scissor_res/survival_2025_01_01/",
  cache_config_path = "Scissor_res/survival_2025_01_01/cache_config.json",
  cache_data = list(result = my_matrix),
  screen_method_config = config
)
WriteCache(cache)

# Force CSV format for small data frames
WriteCache(cache, format = "csv")

## End(Not run)
```

WriteCacheMeta *Write Cache Metadata*

Description

Writes metadata for SigBridgeR cache identification to a JSON file. The metadata includes general system information, cache configuration, and an optional user-provided description.

Usage

```
WriteCacheMeta(cache_config, ...)
```

Arguments

cache_config	A ScreenMethodConfig or ScreenMethodCache S7 object containing the cache configuration to write.
...	Additional arguments passed to methods. For ScreenMethodConfig: • file: character(1). Path to the JSON metadata file to be written. • additional_description: character or NULL. An optional string describing the cache entry. Default: NULL. • verbose: logical. Whether to print progress messages. Default: TRUE. For ScreenMethodCache: • additional_description: character or NULL. An optional string describing the cache entry. Default: NULL. • verbose: logical. Whether to print progress messages. Default: TRUE.

Details

This function creates a JSON metadata file containing:

- `general`: File path, OS information, timestamp, R version, and SigBridgeR version
- `config`: Screen method, phenotype class, label type, and parameters
- `description`: Additional user-provided description

The JSON file includes a header comment indicating it is required for cache identification and should not be modified.

Only for SigBridgeR version >= 3.7.0.

Value

Invisible. Returns the metadata list that was written to the file.

Methods

`WriteCacheMeta` is an S7 generic with methods available for the following classes:

- `ANY`
- `SigBridgeR::ScreenMethodCache`
- `SigBridgeR::ScreenMethodConfig`

See Also

Other cache_config: `CacheSetHere()`, `CacheSysCall()`, `CheckCache()`, `ChooseCache()`, `LoadCache()`, `WriteCache()`

Examples

```
## Not run:
# Write metadata for a ScreenMethodConfig
config <- ScreenMethodConfig(
  method_name = "Scissor",
  param = list(alpha = 0.05)
)
WriteCacheMeta(config, file = "cache/cache_config.json")

# With an additional description
WriteCacheMeta(config,
  file = "cache/cache_config.json",
  additional_description = "Scissor screening for survival phenotype"
)

## End(Not run)
```

Index

∗ **Add_Screen_method**
RegisterScreenMethod, 115
ScreenStrategy, 139
TemplateScreenFunc, 154
ValidateScreenFunc, 155
∗ **Benchmark**
BenchAcc, 10
BenchGeneScore, 15
BenchGSEA, 20
BenchMemory, 24
∗ **ConfigSet**
BenchGeneScoreConfigSet, 19
BenchGSEAConfigSet, 23
BenchMemoryConfigSet, 27
MethodParamSet, 100
∗ **DEGAS**
DoDEGAS, 44
∗ **LP_SGL**
DoLP_SGL, 50
∗ **PIPET**
DoPIPET, 56
∗ **PhenoMapR**
DoPhenoMapR, 53
∗ **Registering**
RegisterAnnoMethod, 114
RegisterScreenMethod, 115
RegisterSeuratMethod, 117
∗ **S7-Classes**
AnnotationMethod, 8
BenchAccConfig, 12
BenchConfig, 13
BenchGeneScoreConfig, 17
BenchGSEAConfig, 22
BenchMemoryConfig, 26
ScreenMethodCache, 135
ScreenMethodConfig, 137
ScreenMethodResult, 138
∗ **SIDISH**
DoSIDISH, 72
∗ **SigBridgeR-Classes**
Method, 99
ScreenMethod, 134
SigBridgeRBase, 150
∗ **Single_Cell_Annotation_Method**
CellTypistAnnotate, 33
DownloadCellTypistModels, 77
ListCellTypistModels, 92
mLLMCelltypeAnnotate, 101
RegisterAnnoMethod, 114
SCAnnotateStrategy, 120
SingleRAnnotate, 151
∗ **TiRank**
DoTiRank, 74
∗ **bulk_preprocessing**
IsCountsMatrix, 86
∗ **cache_config**
CacheSetHere, 29
CacheSysCall, 31
CheckCache, 36
ChooseCache, 39
LoadCache, 94
WriteCache, 157
WriteCacheMeta, 159
∗ **duplicate aggregation**
aggregate-dups, 6
∗ **input_preprocess**
DrawBulkPCA, 78
PhenoMap, 106
PhenoPreProcess, 107
SCPreProcess, 121
SymbolConvert, 153
∗ **result_management**
MergeResult, 96
∗ **scAB**
DoscAB, 58
∗ **scPAS**
DoscPAS, 67
∗ **scPP**

DoscPP, 69
∗ **scissor**
DoScissor, 64
∗ **screen_method**
DoDEGAS, 44
DoLP_SGL, 50
DoPhenoMapR, 53
DoPIPET, 56
DoscAB, 58
DoSCIPAC, 61
DoScissor, 64
DoscPAS, 67
DoscPP, 69
DoSIDISH, 72
DoTiRank, 74
∗ **single_cell_preprocess**
FindRobustElbow, 80
Pattern2Colname, 105
QCPatternDetect, 112
RegisterSeuratMethod, 117
SCAnnotate, 119
SCPreProcess, 121
SCPreProcessStrategy, 124
∗ **visualization_function**
ScreenFractionPlot, 132
ScreenUpset, 140
+.Seurat *(*plus-methods*)*, 110
+.SigBridgeR::ScreenMethodResult *(*plus-methods*)*, 110
[[.SigBridgeR::SigBridgeRBase *(*Property-visitor*)*, 111
$.SigBridgeR::SigBridgeRBase *(*Property-visitor*)*, 111

AddMetaFeature, 4
AddMisc, 5, *97*
aggregate-dups, 6
AggregateDupCols, *6*
AggregateDupCols *(*aggregate-dups*)*, 6
AggregateDupRows, *6*
AggregateDupRows *(*aggregate-dups*)*, 6
AggregateDups, *6*
AggregateDups *(*aggregate-dups*)*, 6
all.equal(), *28*
annotate_cell_types, *104*
AnnotationMethod, 8, *113*
AnnotationMethod(), *13*, *15*, *18*, *23*, *27*, *136*, *138*, *139*
as.character.SigBridgeR::SigBridgeRBase *(*SigBridgeR-S3*)*, 148
as.list.SigBridgeR::SigBridgeRBase *(*SigBridgeR-S3*)*, 148
AUCell::AUCell_buildRankings(), *16*
AUCell::AUCell_calcAUC(), *16*
autoplot-SigBridgeR, 9
autoplot.Seurat *(*autoplot-SigBridgeR*)*, 9
autoplot.SigBridgeR::ScreenMethodResult *(*autoplot-SigBridgeR*)*, 9
AVAILABLE_ANNOTATE_METHODS *(*SCAnnotateStrategy*)*, 120
AVAILABLE_SCREEN_METHODS *(*ScreenStrategy*)*, 139

bench::mark(), *24*, *25*
bench_probe_resources(), *25*, *28*
BenchAcc, 10
BenchAcc(), *12*, *13*, *16*, *21*, *25*
BenchAccConfig, *11*, 12
BenchAccConfig(), *9*, *11*, *15*, *18*, *23*, *27*, *136*, *138*, *139*
BenchConfig, *12*, 13, *17*, *22*, *26*
BenchConfig(), *9*, *13*, *18*, *23*, *27*, *136*, *138*, *139*
BenchGeneScore, 15
BenchGeneScore(), *11*, *17–19*, *21*, *25*
BenchGeneScore.Seurat(), *16*
BenchGeneScoreConfig, *16*, 17
BenchGeneScoreConfig(), *9*, *13*, *15*, *16*, *19*, *23*, *27*, *136*, *138*, *139*
BenchGeneScoreConfigSet, 19
BenchGeneScoreConfigSet(), *16*, *18*, *23*, *29*, *101*
BenchGSEA, 20
BenchGSEA(), *11*, *16*, *22*, *25*
BenchGSEAConfig, *20*, *21*, 22
BenchGSEAConfig(), *9*, *13*, *15*, *18*, *21*, *23*, *27*, *136*, *138*, *139*
BenchGSEAConfigSet, 23
BenchGSEAConfigSet(), *19*, *23*, *29*, *101*
BenchMemory, 24
BenchMemory(), *11*, *16*, *21*, *26–28*
BenchMemoryConfig, *24*, 26
BenchMemoryConfig(), *9*, *13*, *15*, *18*, *23*, *25*, *136*, *138*, *139*
BenchMemoryConfigSet, 27
BenchMemoryConfigSet(), *19*, *23*, *25*, *101*
brewer.pal.info, *10*

CacheSetHere, 29
CacheSetHere(), *31*, *32*, *37*, *40*, *45*, *46*, *60*, *66*, *75*, *76*, *95*, *159*, *160*
CacheSysCall, 31
CacheSysCall(), *30*, *37*, *40*, *95*, *159*, *160*
callr::r_bg(), *24*
case_when, *107*
CellTypistAnnotate, 33, *77*, *92*
CellTypistAnnotate(), *78*, *92*, *104*, *115*, *121*, *152*
CheckCache, 36
CheckCache(), *30–32*, *40*, *95*, *159*, *160*
CheckNA, 37
CheckNA(), *82*
ChooseCache, 39
ChooseCache(), *30*, *32*, *37*, *95*, *159*, *160*
ChooseNormalization, 40
collapse_probes_to_genes, *54*
complete.cases, *38*
CountsToTPM, 43

DEGASEnvSet(), *47*
DEGASModelDetect(), *47*
DEGASParamSet(), *47*
derive_reference_from_bulk, *55*
DiscretePalette, *10*
DoDEGAS, 44
DoDEGAS(), *52*, *56*, *58*, *60*, *63*, *67*, *69*, *71*, *74*, *76*, *126*, *131*
DoDEGASv2, 47
DoDEGASv2(), *126*, *132*
DoLP_SGL, 50
DoLP_SGL(), *47*, *56*, *58*, *60*, *63*, *67*, *69*, *71*, *74*, *76*, *126*, *131*
DoPhenoMapR, 53
DoPhenoMapR(), *47*, *52*, *58*, *60*, *63*, *67*, *69*, *71*, *74*, *76*, *126*, *132*
DoPIPET, 56
DoPIPET(), *47*, *52*, *56*, *60*, *63*, *67*, *69*, *71*, *74*, *76*, *126*, *132*
DoscAB, 58
DoscAB(), *47*, *52*, *56*, *58*, *63*, *67*, *69*, *71*, *74*, *76*, *126*, *131*
DoSCIPAC, 61
DoSCIPAC(), *47*, *52*, *56*, *58*, *60*, *67*, *69*, *71*, *74*, *76*, *126*, *132*
DoScissor, 64
DoScissor(), *47*, *52*, *56*, *58*, *60*, *63*, *69*, *71*, *74*, *76*, *126*, *131*
DoScissorCellEval(), *67*
DoScissorRelTest(), *67*
DoscPAS, 67
DoscPAS(), *47*, *52*, *56*, *58*, *60*, *63*, *67*, *71*, *74*, *76*, *126*, *131*
DoscPP, 69
DoscPP(), *47*, *52*, *56*, *58*, *60*, *63*, *67*, *69*, *74*, *76*, *126*, *131*
DoSIDISH, 72
DoSIDISH(), *47*, *52*, *56*, *58*, *60*, *63*, *67*, *69*, *71*, *76*, *126*, *132*
DoTiRank, 74
DoTiRank(), *47*, *52*, *56*, *58*, *60*, *63*, *67*, *69*, *71*, *74*, *126*, *130*, *132*
DownloadCellTypistModels, 77, *92*
DownloadCellTypistModels(), *35*, *92*, *104*, *115*, *121*, *152*
DrawBulkPCA, 78
DrawBulkPCA(), *107*, *109*, *123*, *154*

fcase, *107*
fgsea::fgsea(), *20*, *21*, *23*
fgsea::fgseaMultilevel(), *20*
fgsea::fgseaSimple(), *20*
FilterTumorCell(), *81*, *106*, *113*, *118*, *120*, *123*, *125*
FindRobustElbow, 80
FindRobustElbow(), *106*, *113*, *118*, *120*, *123*, *125*
format(), *148*, *149*
format.SigBridgeR::SigBridgeRBase *(*SigBridgeR-S3*)*, 148
FPKMToTPM, 81

getFuncOption, 82
GetGeneLength, 84
ggforce::geom_mark_ellipse(), *78*
GSVA::gsva(), *16*, *19*
GSVA::ssgseaParam(), *16*, *19*

interactive_consensus_annotation, *104*
InterceptStrategy, 85
is.na, *38*
IsCountsMatrix, 86
IsSkewedDynamic, 87

length.SigBridgeR::SigBridgeRBase *(*SigBridgeR-S3*)*, 148
list_to_chr, 90

list_to_chr_formated, 91
ListCellTypistModels, 92
ListCellTypistModels(), *35*, *78*, *104*, *115*, *121*, *152*
ListPyEnv, 93
LoadCache, 94
LoadCache(), *30–32*, *37*, *40*, *159, 160*
LoadRefData, 95

MergeResult, 96
MergeResult(), *110*
Method, *8, 9*, 99, *134, 135*
Method(), *135*, *150*
MethodParamSet, 100
MethodParamSet(), *13–15*, *18, 19*, *22, 23*, *27*, *29*
mLLMCelltypeAnnotate, 101
mLLMCelltypeAnnotate(), *35*, *78*, *92*, *115*, *121*, *152*

na.omit, *38*
names.SigBridgeR::SigBridgeRBase *(*SigBridgeR-S3*)*, 148
names<-.SigBridgeR::SigBridgeRBase *(*SigBridgeR-S3*)*, 148

palette_SigBridgeR, 105
patchwork, *78*
Pattern2Colname, 105
Pattern2Colname(), *81*, *113*, *118*, *120*, *123*, *125*
PercentageFeatureSet, *112*
PhenoMap, 106, *107, 108*
PhenoMap(), *79*, *109*, *123*, *154*
PhenoPreProcess, 107
PhenoPreProcess(), *79*, *107*, *123*, *154*
plus-methods, 110
precog, *54*
print.SigBridgeR::SigBridgeRBase *(*SigBridgeR-S3*)*, 148
Property-visitor, 111
property_chr *(*SigBridgeR-properties*)*, 146
property_data_list *(*SigBridgeR-properties*)*, 146
property_fn *(*SigBridgeR-properties*)*, 146
property_list *(*SigBridgeR-properties*)*, 146
property_mapper_fn *(*SigBridgeR-properties*)*, 146
property_matrix *(*SigBridgeR-properties*)*, 146
property_phenotype *(*SigBridgeR-properties*)*, 146
property_phenotype_class *(*SigBridgeR-properties*)*, 146
property_py_env *(*SigBridgeR-properties*)*, 146
property_seurat *(*SigBridgeR-properties*)*, 146
property_sigbridger_verison *(*SigBridgeR-properties*)*, 146
property_verison *(*SigBridgeR-properties*)*, 146

QCFilter, 111
QCPatternDetect, *111*, 112, *123*
QCPatternDetect(), *81*, *106*, *118*, *120*, *123*, *125*

readLines(), *84*
Register, 113, *119*
RegisterAnnoMethod, 114, *119*
RegisterAnnoMethod(), *35*, *78*, *92*, *104*, *117, 118*, *121*, *152*
RegisterScreenMethod, 115
RegisterScreenMethod(), *115*, *118*, *139*, *155, 156*
RegisterSeuratMethod, 117
RegisterSeuratMethod(), *81*, *106*, *113*, *115*, *117*, *120*, *123*, *125*
reticulate::conda_create(), *146*
reticulate::virtualenv_create(), *146*
rlang::check_installed(), *16*, *19*

S7::props(), *150*
SCAnnotate, 119
SCAnnotate(), *81*, *106*, *113*, *118*, *123*, *125*
SCAnnotateStrategy, *35*, *78*, *92*, *104*, *113–115*, 120, *152*
SCIntegrate, *97*
SCIntegrate(), *81*, *106*, *113*, *118*, *120*, *123*, *125*
SCPreProcess, 121
SCPreProcess(), *79*, *81*, *106, 107*, *109*, *113*, *118*, *120*, *125*, *154*

SCPreProcessStrategy, *81*, *106*, *113, 114*, *118*, *120*, *123*, 124
Screen, 125
Screen(), *11, 12*, *14*, *16, 17*, *21, 22*, *27*
ScreenFractionPlot, 132
ScreenFractionPlot(), *141*
ScreenMethod, *15*, *99*, *113*, 134
ScreenMethod(), *99*, *150*
ScreenMethodCache, 135
ScreenMethodCache(), *9*, *13*, *15*, *18*, *23*, *27*, *138, 139*
ScreenMethodConfig, 137
ScreenMethodConfig(), *9*, *13*, *15*, *18*, *23*, *27*, *136*, *139*
ScreenMethodResult, *11*, *15, 16*, *34, 35*, *102*, *104*, *110*, 138, *151, 152*
ScreenMethodResult(), *9*, *13*, *15*, *18*, *23*, *27*, *136*, *138*
ScreenStrategy, *13, 14*, *18*, *22*, *27*, *100*, *113, 114*, *117*, *137*, 139, *155, 156*
ScreenUpset, 140
ScreenUpset(), *133*
setFuncOption, 142
SetThreads, 143
SetThreads(), *29*
SetupPyEnv, 145
Seurat, *12*, *14*, *17*, *22, 23*, *27*, *138*, *148*
Seurat::AddModuleScore(), *16*
Seurat::as.Seurat(), *34*, *102*
Seurat::FindAllMarkers(), *50*
Seurat::FindMarkers(), *20, 21*
SeuratObject::Seurat, *148*
SIDISHEnvSet(), *74*
SigBridgeR-properties, 146
SigBridgeR-S3, 148
SigBridgeRBase, *99*, *136–138*, *149*, 150, *150*
SigBridgeRBase(), *99*, *135*
SingleRAnnotate, 151
SingleRAnnotate(), *35*, *78*, *92*, *104*, *115*, *121*
summary.SigBridgeR::SigBridgeRBase *(*SigBridgeR-S3*)*, 148
SymbolConvert, 153
SymbolConvert(), *79*, *107*, *109*, *123*

tcga, *54*
TemplateScreenFunc, 154
TemplateScreenFunc(), *117*, *139*, *156*
TrainDEGASModel(), *47*
UCell::ScoreSignatures_UCell(), *16*
utils::Rprofmem(), *28*

ValidateDEGASParams(), *47*
ValidateLPSGLParams(), *52*
ValidatePhenoMapRParams(), *56*
ValidatePIPETParams(), *58*
ValidatescABParams(), *60*
ValidateScissorParams(), *67*
ValidatescPASParams(), *69*
ValidatescPPParams(), *71*
ValidateScreenFunc, 155
ValidateScreenFunc(), *117*, *139*, *155*
ValidateSIDISHParams(), *74*
ValidateTiRankParams(), *76*

WeightedVote, *97*, 156
WriteCache, 157
WriteCache(), *30–32*, *37*, *40*, *95*, *160*
WriteCacheMeta, 159
WriteCacheMeta(), *30*, *32*, *37*, *40*, *95*, *158, 159*