Package: TorchDecon (via r-universe)

May 26, 2026

Type Package

Title Deep Learning-Based Cell Type Deconvolution Using torch

Version 1.0.0

Date 2026-01-26

Description A deep neural network ensemble approach for cell type deconvolution of bulk RNA-seq data. TorchDecon uses simulated bulk samples generated from single-cell RNA-seq data to train deep neural networks that predict cell type fractions. This package is an R-native implementation based on the Scaden algorithm, built on the torch framework (LibTorch C++ backend) for GPU acceleration and cross-platform compatibility. Seamlessly integrates with Seurat objects (v4 and v5 compatible).

License MIT + file LICENSE

URL https://github.com/Zaoqu-Liu/TorchDecon

BugReports https://github.com/Zaoqu-Liu/TorchDecon/issues

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.0.0)

Imports torch (>= 0.9.0), Seurat (>= 4.0.0), SeuratObject (>= 4.0.0), Matrix, data.table, future, future.apply, progressr, cli, methods, stats, utils

Suggests testthat (>= 3.0.0), knitr, rmarkdown, ggplot2, Rcpp

Config/testthat/edition 3

VignetteBuilder knitr

Config/pak/sysreqs cmake libglpk-dev make libicu-dev libpng-dev libuv1-dev libxml2-dev libssl-dev python3 zlib1g-dev

Repository https://zaoqu-liu.r-universe.dev

Date/Publication 2026-01-26 02:33:01 UTC

RemoteUrl https://github.com/Zaoqu-Liu/TorchDecon

RemoteRef main

RemoteSha 139bac3eecffa73b3f979e2a96463fa13f6f05f6

Contents

ApplyScaling *Apply Scaling to New Data*

Description

Apply the same scaling transformation to new prediction data.

Usage

```
ApplyScaling(X, scaling_method = "log_min_max")
```

Arguments

X	Numeric matrix (samples x genes) of new data to scale.
scaling_method	Character. Scaling method to use.

Details

This function applies the same scaling approach used during training to new data. For min-max scaling, each sample is scaled independently based on its own min/max.

Value

Scaled matrix.

CalculateAccuracy *Calculate Accuracy*

Description

Calculate fraction of predictions within threshold of truth.

Usage

```
CalculateAccuracy(predictions, truth, threshold = 0.05)
```

Arguments

predictions	Numeric vector or matrix of predictions.
truth	Numeric vector or matrix of true values.
threshold	Numeric. Threshold for accuracy calculation. Default is 0.05.

Value

Accuracy (fraction between 0 and 1).

CalculateCorrelation *Calculate Correlation*

Description

Calculate correlation between predictions and truth.

Usage

```
CalculateCorrelation(predictions, truth, method = "pearson")
```

Arguments

predictions	Numeric vector or matrix of predictions.
truth	Numeric vector or matrix of true values.
method	Character. Correlation method ("pearson" or "spearman"). Default is "pearson".

Value

Correlation coefficient.

CalculateMAE *Calculate MAE*

Description

Calculate Mean Absolute Error between predictions and truth.

Usage

```
CalculateMAE(predictions, truth)
```

Arguments

predictions	Numeric vector or matrix of predictions.
truth	Numeric vector or matrix of true values.

Value

MAE value.

CalculateMRE *Calculate MRE*

Description

Calculate Mean Relative Error between predictions and truth.

Usage

```
CalculateMRE(predictions, truth, epsilon = 1e-06)
```

Arguments

predictions	Numeric vector or matrix of predictions.
truth	Numeric vector or matrix of true values.
epsilon	Numeric. Small value to avoid division by zero. Default is 1e-6.

Value

MRE value.

CalculateRMSE *Calculate RMSE*

Description

Calculate Root Mean Squared Error between predictions and truth.

Usage

```
CalculateRMSE(predictions, truth)
```

Arguments

predictions	Numeric vector or matrix of predictions.
truth	Numeric vector or matrix of true values.

Value

RMSE value.

`CreateTorchDecon` *Create TorchDecon Model*

Description

Create a TorchDecon model object with specified architecture.

Usage

```
CreateTorchDecon(
  n_features,
  n_classes,
  architecture = c("m256", "m512", "m1024", "custom"),
  hidden_units = NULL,
  dropout_rates = NULL,
  device = "auto",
  seed = NULL
)
```

Arguments

`n_features`	Integer. Number of input features (genes).
`n_classes`	Integer. Number of output classes (cell types).
`architecture`	Character. One of "m256", "m512", "m1024", or "custom". Default is "m256".
`hidden_units`	Integer vector. Custom hidden layer sizes (only used if architecture = "custom"). Default is NULL.
`dropout_rates`	Numeric vector. Custom dropout rates (only used if architecture = "custom"). Default is NULL.
`device`	Character. Device to use ("cpu", "cuda", or "auto"). Default is "auto".
`seed`	Integer. Random seed for reproducibility. Default is NULL.

Details

Pre-defined architectures:

- m256: Hidden units 256-128-64-32, no dropout
- m512: Hidden units 512-256-128-64, dropout 0/0.3/0.2/0.1
- m1024: Hidden units 1024-512-256-128, dropout 0/0.6/0.3/0.1

Value

A TorchDeconModel object containing the neural network and metadata.

Examples

```
## Not run:
# Create a model with m256 architecture
model <- CreateTorchDecon(n_features = 5000, n_classes = 10, architecture = "m256")

# Create a custom architecture
model <- CreateTorchDecon(
  n_features = 5000,
  n_classes = 10,
  architecture = "custom",
  hidden_units = c(512, 256, 128, 64),
  dropout_rates = c(0.1, 0.2, 0.1, 0.1)
)

## End(Not run)
```

CreateTorchDeconEnsemble

Create TorchDecon Ensemble

Description

Create an ensemble of three TorchDecon models with different architectures.

Usage

```
CreateTorchDeconEnsemble(n_features, n_classes, device = "auto", seed = NULL)
```

Arguments

n_features	Integer. Number of input features (genes).
n_classes	Integer. Number of output classes (cell types).
device	Character. Device to use. Default is "auto".
seed	Integer. Random seed. Default is NULL.

Value

A TorchDeconEnsemble object containing three models.

Examples

```
## Not run:
ensemble <- CreateTorchDeconEnsemble(n_features = 5000, n_classes = 10)

## End(Not run)
```

`EvaluateDeconvolution` *Evaluate Deconvolution Results*

Description

Comprehensive evaluation of cell type deconvolution predictions against ground truth fractions. Calculates multiple performance metrics including RMSE, MAE, MRE, Pearson correlation, and accuracy at different thresholds.

Usage

```
EvaluateDeconvolution(
  predictions,
  truth,
  by_celltype = TRUE,
  accuracy_thresholds = c(0.01, 0.05, 0.1)
)
```

Arguments

`predictions`	Data frame or matrix of predicted cell type fractions (samples x cell types).
`truth`	Data frame or matrix of true cell type fractions (samples x cell types).
`by_celltype`	Logical. Calculate metrics per cell type. Default is TRUE.
`accuracy_thresholds`	Numeric vector. Thresholds for accuracy calculation. Default is c(0.01, 0.05, 0.1).

Details

Metrics calculated:

- **RMSE**: Root Mean Squared Error
- **MAE**: Mean Absolute Error
- **MRE**: Mean Relative Error (relative to true values)
- **Pearson**: Pearson correlation coefficient
- **Spearman**: Spearman rank correlation
- **Accuracy**: Fraction of predictions within threshold of truth

Value

A list containing:

overall Data frame with overall metrics (RMSE, MAE, MRE, correlation)

by_celltype Data frame with per-celltype metrics (if by_celltype = TRUE)

accuracy Data frame with accuracy at different thresholds

sample_correlations Numeric vector of per-sample correlations

Examples

```
## Not run:
# Evaluate predictions
eval_results <- EvaluateDeconvolution(predictions, true_fractions)

# View overall metrics
print(eval_results$overall)

# View per-celltype metrics
print(eval_results$by_celltype)

## End(Not run)
```

`EvaluatePredictions` *Evaluate Predictions*

Description

Evaluate predicted cell fractions against known ground truth.

Usage

```
EvaluatePredictions(predictions, truth)
```

Arguments

`predictions`	Data frame of predicted fractions (samples x cell types).
`truth`	Data frame of true fractions (same format as predictions).

Value

A list containing:

rmse Root mean squared error overall

mae Mean absolute error overall

correlation Pearson correlation overall

per_celltype Metrics per cell type

per_sample Metrics per sample

Examples

```
## Not run:
# Evaluate on held-out test data
metrics <- EvaluatePredictions(predictions, true_fractions)
print(metrics$rmse)

## End(Not run)
```

ExportSimulation *Export Simulation to Files*

Description

Export a TorchDeconSimulation object to tab-separated files.

Usage

```
ExportSimulation(simulation, output_dir = ".", prefix = "simulation")
```

Arguments

simulation	A TorchDeconSimulation object.
output_dir	Character. Output directory. Default is current directory.
prefix	Character. Prefix for output files. Default is "simulation".

Value

Invisibly returns the output paths.

ExtractCellTypes *Extract Cell Type Labels from Seurat Object*

Description

Extract cell type annotations from a Seurat object.

Usage

```
ExtractCellTypes(object, celltype_col = NULL)
```

Arguments

object	A Seurat object.
celltype_col	Character. Name of the metadata column containing cell types. If NULL, uses active identity (Idents).

Value

A character vector of cell type labels.

Examples

```
## Not run:
# Use active identity
celltypes <- ExtractCellTypes(seurat_obj)

# Use specific metadata column
celltypes <- ExtractCellTypes(seurat_obj, celltype_col = "cell_type")

## End(Not run)
```

`ExtractSeuratData` *Extract Count Matrix from Seurat Object*

Description

Extract the count matrix from a Seurat object, compatible with both v4 and v5. Uses v4 methods by default, falling back to v5 if necessary.

Usage

```
ExtractSeuratData(
  object,
  assay = NULL,
  slot = c("counts", "data", "scale.data"),
  layer = NULL
)
```

Arguments

`object`	A Seurat object.
`assay`	Character. Name of the assay to use. Default is NULL (uses default assay).
`slot`	Character. Slot to extract. One of "counts", "data", or "scale.data". Default is "counts".
`layer`	Character. For Seurat v5, the layer name. Default is NULL.

Details

This function prioritizes Seurat v4 compatibility. For v4 objects, it uses `GetAssayData()` with the slot argument. For v5 objects, it attempts to use the same method first, then falls back to layer-based access if needed.

Value

A matrix (sparse or dense) containing the expression data.

Examples

```
## Not run:
# Extract counts from default assay
counts <- ExtractSeuratData(seurat_obj)

# Extract normalized data from RNA assay
data <- ExtractSeuratData(seurat_obj, assay = "RNA", slot = "data")

## End(Not run)
```

GenerateExampleData *Generate Example Data*

Description

Generate random example data for testing TorchDecon functionality. Creates a simple Seurat object and bulk expression data.

Usage

```
GenerateExampleData(
  n_cells = 500L,
  n_genes = 200L,
  n_celltypes = 5L,
  n_bulk_samples = 20L,
  seed = 42L
)
```

Arguments

n_cells	Integer. Number of cells in scRNA-seq data. Default is 500.
n_genes	Integer. Number of genes. Default is 200.
n_celltypes	Integer. Number of cell types. Default is 5.
n_bulk_samples	Integer. Number of bulk samples. Default is 20.
seed	Integer. Random seed. Default is 42.

Value

A list containing:

seurat A Seurat object with random scRNA-seq data

bulk_data Matrix of bulk expression data (genes x samples)

Examples

```
## Not run:
example_data <- GenerateExampleData()
seurat_obj <- example_data$seurat
bulk_data <- example_data$bulk_data

## End(Not run)
```

`GetTrainingHistory` *Get Training History*

Description

Extract training history from a trained model.

Usage

```
GetTrainingHistory(model)
```

Arguments

`model`	A trained TorchDeconModel or TorchDeconEnsemble object.

Value

A data frame with training loss (and validation loss if applicable).

`LoadModel` *Load TorchDecon Model*

Description

Load a previously saved TorchDecon model or ensemble from disk.

Usage

```
LoadModel(path, device = "auto")
```

Arguments

`path`	Character. Directory path where the model was saved.
`device`	Character. Device to load the model onto ("cpu", "cuda", or "auto"). Default is "auto".

Value

A TorchDeconModel or TorchDeconEnsemble object.

Examples

```
## Not run:
model <- LoadModel("my_model")
predictions <- PredictFractions(model, new_data)

## End(Not run)
```

MergeSimulations *Merge Simulations*

Description

Merge multiple TorchDeconSimulation objects into one.

Usage

```
MergeSimulations(...)
```

Arguments

...	TorchDeconSimulation objects to merge, or a list of them.

Details

The merged object will contain:

- Combined bulk counts (horizontally concatenated)
- Combined cell fractions (vertically concatenated)
- Union of all genes (with NA handling)
- Union of all cell types (with NA/0 for missing types)

Value

A merged TorchDeconSimulation object.

Examples

```
## Not run:
sim1 <- SimulateBulk(seurat1, n_samples = 500)
sim2 <- SimulateBulk(seurat2, n_samples = 500)
merged <- MergeSimulations(sim1, sim2)

## End(Not run)
```

PlotEvaluation *Plot Evaluation Results*

Description

Visualize evaluation metrics for deconvolution results.

Usage

```
PlotEvaluation(
  evaluation,
  type = c("bar", "correlation", "scatter", "heatmap"),
  predictions = NULL,
  truth = NULL
)
```

Arguments

evaluation	A TorchDeconEvaluation object from EvaluateDeconvolution().
type	Character. Type of plot: "correlation", "scatter", "heatmap", or "bar". Default is "bar".
predictions	Data frame of predictions (required for "scatter" and "heatmap").
truth	Data frame of true values (required for "scatter" and "heatmap").

Value

A ggplot2 object.

PlotTrainingHistory *Plot Training History*

Description

Visualize training loss over steps for TorchDecon models.

Usage

```
PlotTrainingHistory(model, log_scale = FALSE, smooth = TRUE, smooth_span = 0.1)
```

Arguments

model	A trained TorchDeconModel or TorchDeconEnsemble object, or a data frame from GetTrainingHistory().
log_scale	Logical. Use log scale for y-axis. Default is FALSE.
smooth	Logical. Add smoothed line. Default is TRUE.
smooth_span	Numeric. Span for LOESS smoothing. Default is 0.1.

Details

This function requires ggplot2 for plotting. If ggplot2 is not available, it will return the training history data frame.

Value

A ggplot2 object (if ggplot2 is available), otherwise NULL.

Examples

```
## Not run:
# Plot training history
PlotTrainingHistory(trained_model)

# With log scale
PlotTrainingHistory(trained_model, log_scale = TRUE)

## End(Not run)
```

PredictFractions *Predict Cell Type Fractions*

Description

Use a trained TorchDecon model or ensemble to predict cell type fractions from bulk RNA-seq data.

Usage

```
PredictFractions(
  model,
  data,
  scaling = "log_min_max",
  return_all = FALSE,
  verbose = TRUE
)
```

Arguments

model	A trained TorchDeconModel or TorchDeconEnsemble object.
data	Matrix, data frame, or file path to bulk RNA-seq data (genes x samples). Can also be a TorchDeconProcessed object.
scaling	Character. Scaling method to apply. Default is "log_min_max". Set to NULL to skip scaling (if data is already processed).
return_all	Logical. For ensemble, return predictions from all models in addition to the average. Default is FALSE.
verbose	Logical. Print progress messages. Default is TRUE.

Details

For ensemble models, predictions are averaged across all three sub-models (m256, m512, m1024) to produce the final prediction.

The input data must contain the same genes used during training. Missing genes will cause an error.

Value

A data frame of predicted cell type fractions with samples as rows and cell types as columns. If return_all = TRUE for ensemble, returns a list with 'average' and individual model predictions.

Examples

```
## Not run:
# Predict with a trained ensemble
predictions <- PredictFractions(trained_ensemble, bulk_data)

# Get individual model predictions
all_predictions <- PredictFractions(trained_ensemble, bulk_data, return_all = TRUE)

## End(Not run)
```

---

print.TorchDeconEnsemble

Print Method for TorchDeconEnsemble

Description

Print summary of a TorchDeconEnsemble object.

Usage

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

Arguments

x	A TorchDeconEnsemble object.
...	Additional arguments (ignored).

print.TorchDeconEvaluation *Print Method for TorchDeconEvaluation*

Description

Print summary of evaluation results.

Usage

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

Arguments

x	A TorchDeconEvaluation object.
...	Additional arguments (ignored).

print.TorchDeconModel *Print Method for TorchDeconModel*

Description

Print summary of a TorchDeconModel object.

Usage

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

Arguments

x	A TorchDeconModel object.
...	Additional arguments (ignored).

print.TorchDeconProcessed

Print Method for TorchDeconProcessed

Description

Print summary of processed data.

Usage

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

Arguments

x	A TorchDeconProcessed object.
...	Additional arguments (ignored).

print.TorchDeconSimulation

Print Method for TorchDeconSimulation

Description

Print summary of simulation results.

Usage

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

Arguments

x	A TorchDeconSimulation object.
...	Additional arguments (ignored).

ProcessPredictionData *Process Prediction Data*

Description

Process bulk RNA-seq data for prediction using an existing TorchDecon model.

Usage

```
ProcessPredictionData(data, genes, scaling = "log_min_max", verbose = TRUE)
```

Arguments

data	Matrix or data frame of bulk expression data (genes x samples).
genes	Character vector of genes to use (signature genes from training).
scaling	Character. Scaling method matching training. Default is "log_min_max".
verbose	Logical. Print progress. Default is TRUE.

Value

Processed matrix (samples x genes) ready for prediction.

ProcessTrainingData *Process Training Data for TorchDecon*

Description

Preprocess simulated bulk data for model training. This includes log transformation, scaling, and gene filtering based on variance and intersection with prediction data.

Usage

```
ProcessTrainingData(
  simulation,
  prediction_data = NULL,
  var_cutoff = 0.1,
  scaling = c("log_min_max", "log_zscore", "none"),
  verbose = TRUE
)
```

Arguments

simulation	A TorchDeconSimulation object from SimulateBulk, or a list/matrix of bulk counts.
prediction_data	Matrix or data frame of bulk expression data for prediction (genes in rows, samples in columns). Used to find common genes.
var_cutoff	Numeric. Filter out genes with variance below this threshold. Default is 0.1.
scaling	Character. Scaling method to use. One of "log_min_max" (default), "log_zscore", or "none".
verbose	Logical. Print progress messages. Default is TRUE.

Details

The preprocessing pipeline:

1. Find common genes between training and prediction data
2. Filter genes by variance threshold
3. Apply log2(x + 1) transformation
4. Apply sample-wise min-max scaling (or z-score)

Value

A list containing:

X Processed expression matrix (samples x genes), ready for training

Y Cell type fractions matrix (samples x cell types)

genes Character vector of genes used (signature genes)

celltypes Character vector of cell type names

scaling Scaling method used

scaling_params Parameters for scaling (for applying to new data)

Examples

```
## Not run:
# Basic processing
processed <- ProcessTrainingData(simulation, prediction_data = bulk_data)

# Custom variance cutoff
processed <- ProcessTrainingData(
  simulation,
  prediction_data = bulk_data,
  var_cutoff = 0.05,
  scaling = "log_min_max"
)

## End(Not run)
```

QuickPredict *Quick Prediction with Pre-trained Model*

Description

Load a pre-trained model and make predictions on new bulk data.

Usage

```
QuickPredict(model_path, bulk_data, output_file = NULL, verbose = TRUE)
```

Arguments

model_path	Character. Path to saved model directory.
bulk_data	Matrix or file path to bulk RNA-seq data.
output_file	Character. Path to save predictions. Default is NULL.
verbose	Logical. Print progress. Default is TRUE.

Value

Data frame of predicted cell fractions.

Examples

```
## Not run:
predictions <- QuickPredict(
  model_path = "trained_model",
  bulk_data = "bulk_expression.txt",
  output_file = "predictions.txt"
)

## End(Not run)
```

RunTorchDecon *Run Complete TorchDecon Workflow*

Description

A convenience function that runs the complete TorchDecon workflow: simulate bulk data, process, train ensemble, and optionally predict.

Usage

```
RunTorchDecon(
  seurat_object,
  bulk_data = NULL,
  celltype_col = NULL,
  assay = NULL,
  n_samples = 1000L,
  cells_per_sample = 100L,
  sparse_fraction = 0.5,
  unknown_celltypes = NULL,
  num_steps = 1000L,
  batch_size = 128L,
  learning_rate = 1e-04,
  validation_split = 0,
  early_stopping = FALSE,
  patience = 100L,
  var_cutoff = 0.1,
  scaling = "log_min_max",
  model_type = c("ensemble", "single"),
  architecture = c("m512", "m256", "m1024"),
  device = "auto",
  save_model = NULL,
  seed = 42L,
  verbose = TRUE,
  n_cores = 1L
)
```

Arguments

seurat_object	A Seurat object with cell type annotations.
bulk_data	Matrix of bulk RNA-seq data for prediction (genes x samples). If NULL, only training is performed.
celltype_col	Character. Metadata column with cell type labels.
assay	Character. Assay to use from Seurat object. Default is NULL (default assay).
n_samples	Integer. Number of bulk samples to simulate. Default is 1000.
cells_per_sample	Integer. Cells per simulated sample. Default is 100.
sparse_fraction	Numeric. Fraction of sparse samples (0-1). Default is 0.5.
unknown_celltypes	Character vector. Cell types to merge into "Unknown". Default is NULL.
num_steps	Integer. Training steps per model. Default is 1000 (matches Python).
batch_size	Integer. Training batch size. Default is 128.
learning_rate	Numeric. Learning rate. Default is 0.0001.
validation_split	Numeric. Fraction for validation (0-1). Default is 0.

early_stopping	Logical. Enable early stopping. Default is FALSE.
patience	Integer. Early stopping patience. Default is 100.
var_cutoff	Numeric. Variance cutoff for gene filtering. Default is 0.1.
scaling	Character. Scaling method: "log_min_max", "log_zscore", or "none". Default is "log_min_max".
model_type	Character. "ensemble" or "single". Default is "ensemble".
architecture	Character. Architecture for single model: "m256", "m512", "m1024". Default is "m512".
device	Character. "auto", "cpu", or "cuda". Default is "auto".
save_model	Character. Path to save trained model. Default is NULL (don't save).
seed	Integer. Random seed. Default is 42.
verbose	Logical. Print progress. Default is TRUE.
n_cores	Integer. Cores for parallel simulation. Default is 1.

Value

A list containing:

model The trained TorchDeconModel or TorchDeconEnsemble

predictions Predicted cell fractions (if bulk_data provided)

simulation The simulation object

processed The processed training data

Examples

```
## Not run:
# Complete workflow with ensemble (default)
result <- RunTorchDecon(
  seurat_object = my_seurat,
  bulk_data = bulk_expression,
  celltype_col = "cell_type",
  n_samples = 2000,
  num_steps = 1000
)

# Single model with early stopping
result <- RunTorchDecon(
  seurat_object = my_seurat,
  bulk_data = bulk_expression,
  model_type = "single",
  architecture = "m1024",
  validation_split = 0.1,
  early_stopping = TRUE
)

# Get predictions
predictions <- result$predictions
```

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

`SaveModel` *Save TorchDecon Model*

Description

Save a trained TorchDecon model or ensemble to disk.

Usage

```
SaveModel(model, path, overwrite = FALSE)
```

Arguments

`model`	A TorchDeconModel or TorchDeconEnsemble object.
`path`	Character. Directory path to save the model.
`overwrite`	Logical. Overwrite existing files. Default is FALSE.

Details

The function saves:

- Network weights (.pt files)
- Model metadata (genes, cell types, architecture)
- Training history (if available)

Value

Invisibly returns the save path.

Examples

```
## Not run:
SaveModel(trained_model, "my_model")

## End(Not run)
```

`seurat-utils` *Seurat Compatibility Utilities*

Description

Utility functions for Seurat v4/v5 compatibility.

SimulateBulk *Simulate Bulk RNA-seq Data from Single-Cell Data*

Description

Generate simulated bulk RNA-seq samples from single-cell RNA-seq data stored in a Seurat object. This function creates artificial bulk samples by randomly sampling and aggregating single cells with known cell type proportions.

Usage

```
SimulateBulk(
  object,
  n_samples = 1000L,
  cells_per_sample = 100L,
  celltype_col = NULL,
  assay = NULL,
  unknown_celltypes = NULL,
  sparse_fraction = 0.5,
  min_celltypes = 1L,
  seed = NULL,
  verbose = TRUE,
  n_cores = 1L
)
```

Arguments

object	A Seurat object containing single-cell RNA-seq data.
n_samples	Integer. Number of bulk samples to simulate. Default is 1000.
cells_per_sample	Integer. Number of cells to aggregate per sample. Default is 100.
celltype_col	Character. Name of metadata column containing cell type labels. If NULL, uses active identity (Idents). Default is NULL.
assay	Character. Name of assay to use. Default is NULL (uses default assay).
unknown_celltypes	Character vector. Cell types to merge into "Unknown" category. Default is NULL (no merging).
sparse_fraction	Numeric. Fraction of samples that should be "sparse" (missing some cell types). Value between 0 and 1. Default is 0.5.
min_celltypes	Integer. Minimum number of cell types in sparse samples. Default is 1.
seed	Integer. Random seed for reproducibility. Default is NULL.
verbose	Logical. Print progress messages. Default is TRUE.
n_cores	Integer. Number of cores for parallel processing. Default is 1.

Details

The simulation process:

1. Generate random cell type fractions that sum to 1
2. Sample cells according to these fractions
3. Sum expression values across sampled cells
4. Create both "normal" (all cell types) and "sparse" (subset of cell types) samples

Value

A list containing:

bulk_counts Matrix of simulated bulk expression (genes x samples)

cell_fractions Data frame of true cell type fractions (samples x cell types)

celltypes Character vector of cell type names

genes Character vector of gene names

metadata List of simulation parameters

Examples

```
## Not run:
# Basic simulation
sim_data <- SimulateBulk(seurat_obj, n_samples = 1000)

# Custom simulation with specific parameters
sim_data <- SimulateBulk(
  seurat_obj,
  n_samples = 2000,
  cells_per_sample = 200,
  celltype_col = "cell_annotation",
  sparse_fraction = 0.3,
  seed = 42
)

## End(Not run)
```

SimulationToDataFrame *Convert Simulation to Data Frame*

Description

Convert a TorchDeconSimulation object to a data frame for export.

Usage

```
SimulationToDataFrame(simulation, what = c("both", "counts", "fractions"))
```

Arguments

simulation	A TorchDeconSimulation object.
what	Character. What to export: "counts", "fractions", or "both". Default is "both".

Value

A data frame or list of data frames.

---

summary.TorchDeconEvaluation

Summary Method for TorchDeconEvaluation

Description

Return summary statistics of evaluation.

Usage

```
## S3 method for class 'TorchDeconEvaluation'
summary(object, ...)
```

Arguments

object	A TorchDeconEvaluation object.
...	Additional arguments (ignored).

Value

Data frame with summary statistics.

---

summary.TorchDeconModel

Summary Method for TorchDeconModel

Description

Print detailed summary of a TorchDeconModel object.

Usage

```
## S3 method for class 'TorchDeconModel'
summary(object, ...)
```

Arguments

object	A TorchDeconModel object.
...	Additional arguments (ignored).

`TrainModel` *Train TorchDecon Model*

Description

Train a TorchDecon model or ensemble on processed training data.

Usage

```
TrainModel(
  model,
  data,
  batch_size = 128L,
  learning_rate = 1e-04,
  num_steps = 1000L,
  validation_split = 0,
  early_stopping = FALSE,
  patience = 500L,
  checkpoint_dir = NULL,
  verbose = TRUE,
  seed = NULL
)
```

Arguments

`model`	A TorchDeconModel or TorchDeconEnsemble object.
`data`	A TorchDeconProcessed object from `ProcessTrainingData`, or a list containing X (features) and Y (labels) matrices.
`batch_size`	Integer. Batch size for training. Default is 128.
`learning_rate`	Numeric. Learning rate for Adam optimizer. Default is 0.0001.
`num_steps`	Integer. Number of training steps. Default is 5000.
`validation_split`	Numeric. Fraction of data to use for validation (0-1). Default is 0 (no validation).
`early_stopping`	Logical. Enable early stopping based on validation loss. Default is FALSE.
`patience`	Integer. Number of steps without improvement before stopping. Default is 500.
`checkpoint_dir`	Character. Directory to save model checkpoints. Default is NULL.
`verbose`	Logical. Print training progress. Default is TRUE.
`seed`	Integer. Random seed. Default is NULL.

Details

The training process uses:

- Adam optimizer with configurable learning rate
- Mean Squared Error (MSE) loss function
- Mini-batch gradient descent
- Optional validation and early stopping

For ensemble models, each sub-model is trained sequentially.

Value

The trained model object (modified in place and returned).

Examples

```
## Not run:
# Train a single model
model <- CreateTorchDecon(n_features = 5000, n_classes = 10)
model <- TrainModel(model, processed_data, num_steps = 5000)

# Train an ensemble
ensemble <- CreateTorchDeconEnsemble(n_features = 5000, n_classes = 10)
ensemble <- TrainModel(ensemble, processed_data, num_steps = 5000)

## End(Not run)
```

Index