

# Package: immLynx (via r-universe)

May 30, 2026

**Title** Linking Advanced TCR Python Pipelines and Hugging Face Models in R

**Version** 1.0.0

**Description** A comprehensive toolkit that bridges popular Python-based immune repertoire analysis tools and Hugging Face protein language models into the R environment. Provides unified interfaces for TCR distance calculations (tcrdist3), sequence generation probability (OLGA), selection inference (soNNia), clustering (clusTCR), protein embeddings (ESM-2), metaclone discovery (metaclonotypist). Fully compatible with the scRepertoire and immApex ecosystem for single-cell immune repertoire analysis.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**biocViews** Software, ImmunoOncology, SingleCell, Classification, Annotation, Sequencing, MotifAnnotation, Clustering, DimensionReduction

**Depends** R (>= 4.5.0)

**Imports** basilisk (>= 1.8.0), reticulate (>= 1.24), immApex, methods, S4Vectors, SingleCellExperiment, stats, SummarizedExperiment, utils

**Suggests** BiocStyle, ggplot2, knitr, markdown, rmarkdown, scater, scran, scRepertoire, spelling, testthat (>= 3.0.0), withr

**VignetteBuilder** knitr

**Language** en-US

**URL** <https://github.com/BorchLab/immLynx/>

**BugReports** <https://github.com/BorchLab/immLynx/issues>

**StagedInstall** no

**Config/testthat/edition** 3

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

**Repository** <https://bioc-release.r-universe.dev>

**Date/Publication** 2026-04-28 13:07:11 UTC

**RemoteUrl** <https://github.com/bioc/immLynx>

**RemoteRef** RELEASE\_3\_23

**RemoteSha** e2fd5794bff0b71d8be08f0154e75395c2262b4a

## Contents

convertToTcrdist . . . . .	2
extractTCRdata . . . . .	3
generateOLGA . . . . .	5
huggingModel . . . . .	5
immLynx_example . . . . .	6
proteinEmbeddings . . . . .	7
runClustTCR . . . . .	9
runEmbeddings . . . . .	10
runHLAassociation . . . . .	12
runMetaclonotypist . . . . .	13
runOLGA . . . . .	15
runSoNNia . . . . .	16
runTCRdist . . . . .	17
summarizeTCRrepertoire . . . . .	19
TCR_summary-class . . . . .	20
tokenizeSequences . . . . .	21
validateTCRdata . . . . .	22
<b>Index</b>	<b>24</b>

---

convertToTcrdist	<i>Convert TCR Data to tcrdist3 Format</i>
------------------	--

---

## Description

Converts TCR data from immLynx/scRepertoire format to the format required by tcrdist3 for distance calculations.

## Usage

```
convertToTcrdist(
  tcr_data,
  chains = c("beta", "alpha", "both"),
  include_count = TRUE
)
```

**Arguments**

**tcr\_data** A data.frame from extractTCRdata() or similar source

**chains** Which chains to include: "alpha", "beta", or "both". Default is "beta".

**include\_count** Logical. If TRUE, adds a 'count' column (default 1). Default is TRUE.

**Value**

A data.frame in tcrdist3 format with columns:

**count** Clone count (default 1)

**v\_a\_gene** Alpha V gene (if alpha chain included)

**j\_a\_gene** Alpha J gene (if alpha chain included)

**cdr3\_a\_aa** Alpha CDR3 amino acid sequence (if alpha chain included)

**v\_b\_gene** Beta V gene (if beta chain included)

**j\_b\_gene** Beta J gene (if beta chain included)

**cdr3\_b\_aa** Beta CDR3 amino acid sequence (if beta chain included)

**Examples**

```
# Convert long-format TCR data to tcrdist3 format
tcr_data <- data.frame(
  barcode = paste0("cell_", 1:5),
  cdr3_aa = c("CASSLGTGELFF", "CASSIRSSYEQYF", "CASSYSTGELFF",
             "CASNQLNEKLFF", "CASSLDRNEQFF"),
  v = paste0("TRBV", c("7-2", "12-3", "5-1", "28", "7-9")),
  j = paste0("TRBJ", c("2-2", "1-1", "2-7", "1-5", "2-1")),
  chain = rep("TRB", 5),
  stringsAsFactors = FALSE
)
tcrdist_format <- convertToTcrdist(tcr_data, chains = "beta")
head(tcrdist_format)
```

---

extractTCRdata

*Extract TCR Data from SingleCellExperiment Object*

---

**Description**

Extracts T-cell receptor data from a SingleCellExperiment object that has been processed with scRepertoire. This is a convenience wrapper around immApex::getIR() that provides additional formatting options.

**Usage**

```
extractTCRdata(
  input,
  chains = c("TRB", "TRA", "TRG", "TRD", "IGH", "IGL", "IGK", "both"),
  format = c("long", "wide"),
  remove_na = TRUE
)
```

**Arguments**

<code>input</code>	A SingleCellExperiment object containing scRepertoire TCR data in metadata.
<code>chains</code>	Which chains to extract: "TRA", "TRB", "TRG", "TRD", "IGH", "IGL", "IGK", or "both" (for TRA and TRB). Default is "TRB".
<code>format</code>	Output format: "long" (one row per chain) or "wide" (one row per cell with columns for each chain). Default is "long".
<code>remove_na</code>	Logical. If TRUE, removes rows with NA CDR3 sequences. Default is TRUE.

**Value**

A data.frame containing:

- barcode** Cell barcode
- cdr3\_aa** CDR3 amino acid sequence
- cdr3\_nt** CDR3 nucleotide sequence (if available)
- v** V gene
- d** D gene (if applicable)
- j** J gene
- c** C gene (if available)
- chain** Chain type (TRA, TRB, etc.)

**Examples**

```
# Extract beta chain data
data(immLynx_example)
tcr_data <- extractTCRdata(immLynx_example, chains = "TRB")
head(tcr_data)
```

---

`generateOLGA`*Generate Random TCR Sequences using OLGA*

---

**Description**

Generate random TCR sequences from OLGA's generative model.

**Usage**

```
generateOLGA(n = 100, model = "humanTRB")
```

**Arguments**

`n` Number of sequences to generate.  
`model` OLGA model to use: "humanTRB", "humanTRA", "humanIGH", "mouseTRB".

**Value**

Data.frame with generated sequences (nt\_seq, aa\_seq, v\_index, j\_index).

**Examples**

```
# Available models
models <- c("humanTRB", "humanTRA", "humanIGH", "mouseTRB")

# Generate 100 random human TRB sequences
random_seqs <- generateOLGA(n = 100, model = "humanTRB")
head(random_seqs)

# Generate human TRA sequences
tra_seqs <- generateOLGA(n = 50, model = "humanTRA")

# Generate mouse TRB sequences
mouse_seqs <- generateOLGA(n = 200, model = "mouseTRB")
```

---

`huggingModel`*Initialize a Hugging Face Model and Tokenizer*

---

**Description**

Downloads or loads a cached pre-trained model and its corresponding tokenizer from the Hugging Face Hub. Uses the basilisk-managed Python environment which includes transformers and torch.

**Usage**

```
huggingModel(model_name = "facebook/esm2_t12_35M_UR50D")
```

**Arguments**

`model_name` A string specifying the model identifier from the Hugging Face Hub. For ESM-2 35M, this is "facebook/esm2\_t12\_35M\_UR50D". Other options include "facebook/esm2\_t33\_650M\_UR50D" for larger models.

**Value**

A list containing the R-wrapped Python objects for the 'model' and 'tokenizer', along with the basilisk process handle. The process must be stopped when no longer needed via `basilisk::basiliskStop()`.

**See Also**

[tokenizeSequences](#), [proteinEmbeddings](#), [runEmbeddings](#)

**Examples**

```
# Default model is ESM-2 35M
model_name <- "facebook/esm2_t12_35M_UR50D"

# Load the default ESM-2 35M model
hf_components <- huggingModel(model_name)
names(hf_components) # "model", "tokenizer", "proc"

# Use with tokenizeSequences and proteinEmbeddings
sequences <- c("CASSLGTGELFF", "CASSIRSSYEQYF")
tokenized <- tokenizeSequences(hf_components$tokenizer,
                               sequences)
embeddings <- proteinEmbeddings(hf_components$model,
                                tokenized,
                                pool = "mean",
                                chunk_size = 32)

# Clean up the basilisk process when done
basilisk::basiliskStop(hf_components$proc)
```

**Description**

An SCE object containing single-cell RNA-seq data from multiple patients with integrated T-cell receptor (TCR) repertoire data from scRepertoire. This dataset is useful for demonstrating the functionality of immLynx analysis functions.

**Usage**

```
data(immlynx_example)
```

**Format**

An SCE object with the following components:

**Assays** RNA expression data

**Metadata** Cell-level metadata including:

- `orig.ident`: Original sample identifier (e.g., "P17B", "P17L")
- `nCount_RNA`: Total RNA counts per cell
- `nFeature_RNA`: Number of detected genes per cell
- `CTgene`: TCR gene information (V/J genes)
- `CTnt`: TCR CDR3 nucleotide sequences
- `CTaa`: TCR CDR3 amino acid sequences
- `CTstrict`: Strict TCR identifier combining gene and sequence
- `clonalFrequency`: Number of cells sharing the same TCR
- `clonalProportion`: Proportion of cells with the same TCR
- `cloneSize`: Categorical clone size (Small, Medium, Large, Hyperexpanded)
- `Patient`: Patient identifier (P17, P18, P19, P20)
- `Type`: Sample type (B = Blood, L = Lymph node)
- `clusters`: Cell cluster assignments

**Details**

This dataset was created by combining 10X Genomics single-cell gene expression and VDJ sequencing data from 8 samples across 4 patients. Each patient contributed two samples: blood (B) and lymph node (L) tissue. More information on the data can be found in the following [manuscript](<https://pubmed.ncbi.nlm.nih.gov/33622974/>).

**Examples**

```
data(immlynx_example)
immlynx_example
```

---

proteinEmbeddings

*Get Protein Embeddings from a Model*

---

**Description**

Applies a pre-trained model to a batch of tokenized sequences to generate embeddings. This is the core embedding function used by [runEmbeddings](#).

**Usage**

```
proteinEmbeddings(
  model,
  tokenized.batch,
  pool = c("none", "mean", "cls"),
  chunk_size = NULL,
  prefer_dtype = c("float16", "bfloat16", "float32"),
  prefer_device = c("auto", "cuda", "mps", "cpu")
)
```

**Arguments**

model	HF model (from AutoModel or similar), typically obtained via <a href="#">huggingModel</a> .
tokenized.batch	A <i>*list*</i> of tokenized tensors OR a list of such lists (i.e., already minibatched). If you pass a single big batch, set chunk_size. Typically obtained via <a href="#">tokenizeSequences</a> .
pool	One of "mean", "cls", or "none". "mean" is recommended for sequence-level embeddings.
chunk_size	If tokenized.batch is a single big batch, split it into chunks of this many sequences. Ignored if you pre-batched upstream.
prefer_dtype	One of "float16", "bfloat16", "float32". Lower precision uses less memory but may reduce accuracy.
prefer_device	One of "auto", "cuda", "mps", "cpu". "auto" will select the best available device.

**Value**

An R matrix [n\_sequences x hidden] if pool != "none". If pool == "none", returns a list of arrays per chunk.

**See Also**

[runEmbeddings](#) for a higher-level wrapper that works directly with SingleCellExperiment objects.

**Examples**

```
sequences <- c("CASSLGTGELFF", "CASSIRSSYEQYF", "CASSYSTGELFF")

# Full workflow: load model, tokenize, embed
hf_components <- huggingModel()
tokenized <- tokenizeSequences(hf_components$tokenizer,
                              sequences)

# Mean pooling (recommended for sequence-level tasks)
embeddings <- proteinEmbeddings(hf_components$model,
                                tokenized,
                                pool = "mean",
                                chunk_size = 32)
dim(embeddings) # [n_sequences x hidden_dim]
```

```

# CLS token embedding
cls_emb <- proteinEmbeddings(hf_components$model,
                             tokenized,
                             pool = "cls",
                             chunk_size = 32)

# Per-token embeddings (no pooling)
token_emb <- proteinEmbeddings(hf_components$model,
                               tokenized,
                               pool = "none",
                               chunk_size = 32)

# Use GPU with half precision for speed
embeddings_gpu <- proteinEmbeddings(
  hf_components$model, tokenized,
  pool = "mean", chunk_size = 64,
  prefer_device = "cuda",
  prefer_dtype = "float16")

```

---

runClustTCR

*Run clusTCR Clustering on scRepertoire Data*


---

### Description

This function extracts TCR sequences from a SingleCellExperiment object with scRepertoire data and performs clustering using the clusTCR algorithm.

### Usage

```

runClustTCR(
  input,
  chains = c("TRB", "TRA", "both"),
  method = "mcl",
  combine_chains = FALSE,
  return_object = TRUE,
  column_prefix = "clustcr",
  ...
)

```

### Arguments

input	A SingleCellExperiment object containing scRepertoire TCR data in the meta-data.
chains	Character string specifying which chains to use: "TRA", "TRB", or "both". Default is "TRB".
method	Clustering method passed to clusTCR. Default is "mcl" (Markov Clustering), which is accurate for typical repertoire datasets.

**combine\_chains** Logical. If TRUE and chains="both", concatenates alpha and beta sequences with "\_". Default is FALSE (clusters chains separately).  
**return\_object** Logical. If TRUE, adds cluster assignments back to the input object metadata. If FALSE, returns only the clustering results. Default is TRUE.  
**column\_prefix** Prefix for the new metadata column(s). Default is "cluster".  
 ... Additional arguments passed to calculate.cluster (e.g., inflation).

### Value

If return\_object=TRUE, returns the input object with cluster assignments added to metadata. If return\_object=FALSE, returns a data.frame with barcodes and cluster assignments.

### Examples

```

data(immLynx_example)

# Cluster TRB chain using MCL algorithm
sce <- runClustTCR(immLynx_example,
                  chains = "TRB")

# Adjust MCL inflation parameter
sce <- runClustTCR(immLynx_example,
                  chains = "TRB",
                  inflation = 3.0)

# Cluster both chains separately
sce <- runClustTCR(immLynx_example,
                  chains = "both")

# Combine alpha and beta chains before clustering
sce <- runClustTCR(immLynx_example,
                  chains = "both",
                  combine_chains = TRUE)

# Get results as data.frame
clusters_df <- runClustTCR(immLynx_example,
                          chains = "TRB",
                          return_object = FALSE)

```

---

runEmbeddings

*Generate Protein Language Model Embeddings for TCR Sequences*

---

### Description

Extracts TCR CDR3 sequences from a SingleCellExperiment object and generates embeddings using a protein language model (e.g., ESM-2).

**Usage**

```
runEmbeddings(
  input,
  chains = c("TRB", "TRA", "both"),
  model_name = "facebook/esm2_t12_35M_UR50D",
  pool = "mean",
  chunk_size = 32,
  reduction_name = "tcr_esm",
  reduction_key = "ESM_",
  return_object = TRUE,
  ...
)
```

**Arguments**

input	A SingleCellExperiment object containing scRepertoire TCR data.
chains	Which chain(s) to embed: "TRB", "TRA", or "both". Default is "TRB".
model_name	Hugging Face model name. Default is "facebook/esm2_t12_35M_UR50D". Other options: "facebook/esm2_t33_650M_UR50D", "facebook/esm2_t36_3B_UR50D"
pool	Pooling method: "mean", "cls", or "none". Default is "mean".
chunk_size	Number of sequences to process at once. Default is 32.
reduction_name	Name for the dimensional reduction. Default is "tcr_esm".
reduction_key	Key prefix for embeddings in reduction. Default is "ESM_".
return_object	Logical. If TRUE, adds embeddings as dimensional reduction. If FALSE, returns list with embeddings and metadata. Default is TRUE.
...	Additional arguments passed to proteinEmbeddings().

**Details**

This function uses protein language models to generate dense vector representations of TCR CDR3 sequences. These embeddings can be used for: - Dimensionality reduction and visualization - Clustering TCRs by sequence similarity - Downstream machine learning tasks

**Value**

If return\_object=TRUE, returns input object with embeddings added as a dimensional reduction. If FALSE, returns list with embeddings matrix and metadata.

**Examples**

```
data(immlynx_example)

# Generate ESM-2 embeddings for TRB chain
sce <- runEmbeddings(immlynx_example,
  chains = "TRB")

# Use a larger ESM-2 model
```

```
sce <- runEmbeddings(immLynx_example,
                    chains = "TRB",
                    model_name = "facebook/esm2_t33_650M_UR50D")

# Embed both chains together
sce <- runEmbeddings(immLynx_example,
                    chains = "both")

# Use CLS pooling instead of mean pooling
sce <- runEmbeddings(immLynx_example,
                    chains = "TRB",
                    pool = "cls")

# Get raw embeddings as a list
emb_list <- runEmbeddings(immLynx_example,
                        chains = "TRB",
                        return_object = FALSE)

dim(emb_list$embeddings)
```

---

runHLAassociation

*Perform HLA Association Analysis on Metaclones*


---

## Description

Tests associations between metacclone membership and HLA alleles using Fisher's exact test with FDR correction.

## Usage

```
runHLAassociation(
  metacclone_data,
  hla_data,
  by = "barcode",
  fdr_threshold = 0.05
)
```

## Arguments

metacclone_data	A data.frame with metacclone assignments, typically from runMetaclonotypist() with return_input=FALSE.
hla_data	A data.frame with HLA typing information. Must have a 'barcode' or 'sample' column for matching and columns for each HLA allele.
by	Column name to use for matching between datasets. Default is "barcode".
fdr_threshold	FDR threshold for significance. Default is 0.05.

**Value**

A data.frame with HLA association results:

**metaclone** Metaclone identifier  
**hla\_allele** HLA allele tested  
**odds\_ratio** Association odds ratio  
**pvalue** Raw p-value from Fisher's exact test  
**fdr** FDR-adjusted p-value  
**significant** Whether FDR < threshold

**Examples**

```
# Create example metaclone and HLA data
metaclone_data <- data.frame(
  barcode = paste0("cell_", 1:20),
  metaclone = rep(c("MC1", "MC2"), each = 10),
  stringsAsFactors = FALSE
)
hla_data <- data.frame(
  barcode = paste0("cell_", 1:20),
  HLA_A = c(rep("A*02:01", 8), rep("A*01:01", 4),
            rep("A*02:01", 3), rep("A*01:01", 5)),
  stringsAsFactors = FALSE
)
results <- runHLAassociation(metaclone_data, hla_data)
```

---

runMetaclonotypist      *Run Metaclonotypist for TCR Metaclone Discovery*

---

**Description**

Identifies TCR metaclones (groups of related T cell receptors) using the metaclonotypist pipeline. Metaclonotypist uses a two-stage approach: fast edit-distance-based screening followed by TCRdist or SCEPTR refinement.

**Usage**

```
runMetaclonotypist(
  input,
  chains = c("beta", "alpha"),
  method = c("tcrdist", "scepter"),
  max_edits = 2,
  max_dist = NULL,
  clustering = c("cc", "leiden", "louvain", "mcl"),
  resolution = 1,
  return_input = TRUE,
  column_name = "metaclone"
)
```

**Arguments**

input	A SingleCellExperiment object with scRepertoire data, or a data.frame with TCR data.
chains	Which chain to analyze: "alpha" or "beta". Default is "beta".
method	Distance metric for refinement: "tcrdist" (default) or "scepttr".
max_edits	Maximum CDR3 edit distance for initial screening. Default is 2.
max_dist	Maximum distance threshold for clustering. Default is 20 for tcrdist, 1.5 for scepttr.
clustering	Clustering algorithm: "cc" (connected components, default), "leiden", "louvain", or "mcl".
resolution	Resolution parameter for leiden/louvain clustering. Default is 1.0.
return_input	Logical. If TRUE and input is a SingleCellExperiment object, adds metaclone assignments to metadata. Default is TRUE.
column_name	Name for the metadata column. Default is "metaclone".

**Details**

Metaclonotypist identifies groups of related TCRs that may recognize similar antigens. The algorithm: 1. Uses the Symdel algorithm for fast edit-distance-based candidate identification 2. Refines candidates using TCRdist or SCEPTR similarity metrics 3. Applies graph-based clustering (Leiden by default) to identify metaclones

**Value**

If return\_input=TRUE and input is a SingleCellExperiment, returns the object with metaclone assignments added to metadata. Otherwise returns a data.frame with:

**barcode** Cell barcode  
**cdr3\_aa** CDR3 amino acid sequence  
**metaclone** Metaclone cluster assignment  
**metaclone\_size** Number of cells in the metaclone

**References**

Metaclonotypist: <https://github.com/qimmuno/metaclonotypist>

**Examples**

```
data(immLynx_example)

# Run metaclonotypist on beta chain
sce <- runMetaclonotypist(immLynx_example, chains = "beta")

# Get results as data.frame instead of adding to object
metaclones <- runMetaclonotypist(immLynx_example,
                                return_input = FALSE)
```

```
# Adjust edit distance threshold
sce <- runMetaclonotypist(immlynx_example,
                          max_edits = 3,
                          max_dist = 50)
```

---

runOLGA	<i>Calculate Generation Probability (Pgen) for TCRs in scRepertoire Data</i>
---------	--

---

### Description

Extracts TCR sequences from a SingleCellExperiment object and calculates their generation probability using OLGA.

### Usage

```
runOLGA(
  input,
  chains = c("TRB", "TRA"),
  model = NULL,
  organism = "human",
  use_vj_genes = FALSE,
  return_object = TRUE,
  column_name = "olga_pgen"
)
```

### Arguments

input	A SingleCellExperiment object containing scRepertoire TCR data.
chains	Which chain to analyze: "TRA" or "TRB". Default is "TRB".
model	OLGA model to use. Options: "humanTRB", "humanTRA", "humanIGH", "mouseTRB". If NULL, will be inferred from organism and chains parameters.
organism	Organism: "human" or "mouse". Used if model is NULL. Default is "human".
use_vj_genes	Logical. If TRUE, includes V and J gene information in Pgen calculation. Default is FALSE (sequence-only Pgen).
return_object	Logical. If TRUE, adds Pgen values to metadata. Default is TRUE.
column_name	Name for the metadata column. Default is "olga_pgen".

### Value

If return\_object=TRUE, returns input object with Pgen added to metadata. If FALSE, returns data.frame with barcodes, sequences, and Pgen values.

**Examples**

```

data(immLynx_example)

# Calculate Pgen for TRB chain
sce <- runOLGA(immLynx_example, chains = "TRB")

# Calculate Pgen for TRA chain
sce <- runOLGA(immLynx_example, chains = "TRA")

# Include V and J gene information
sce <- runOLGA(immLynx_example,
               chains = "TRB",
               use_vj_genes = TRUE)

# Get results as data.frame
pgen_df <- runOLGA(immLynx_example,
                  chains = "TRB",
                  return_object = FALSE)

# Specify model explicitly for mouse data
sce <- runOLGA(immLynx_example,
               model = "mouseTRB")

```

---

runSoNNia

*Run soNNia Selection Analysis*


---

**Description**

Infer selection pressures on TCRs using soNNia. Requires a background dataset of unselected sequences (generated by OLGA).

**Usage**

```

runSoNNia(
  input,
  chains = c("TRB", "TRA"),
  background_file,
  organism = "human",
  save_folder = "sonia_output",
  n_epochs = 100,
  return_object = TRUE
)

```

**Arguments**

input	A SingleCellExperiment object containing scRepertoire TCR data.
chains	Which chain to analyze: "TRB" or "TRA". Default is "TRB".

background\_file Path to CSV file with background sequences (from generateOLGA).  
 organism Organism: "human" or "mouse". Default is "human".  
 save\_folder Directory to save soNNia model. Default is "sonia\_output".  
 n\_epochs Number of training epochs. Default is 100.  
 return\_object If TRUE, adds selection scores to metadata. Default is TRUE.

### Details

This function requires a background dataset of unselected TCR sequences, which can be generated using generateOLGA(). The selected sequences are extracted from the input object and compared to this background to infer selection pressures.

### Value

If return\_object=TRUE, returns input object with selection scores in metadata. Otherwise returns the soNNia results.

### Examples

```
data(immLynx_example)
## Not run:
# Step 1: Generate background sequences with OLGA
background <- generateOLGA(n = 1000, model = "humanTRB")
bg_file <- tempfile(fileext = ".csv")
utils::write.csv(background, bg_file, row.names = FALSE)

# Step 2: Run soNNia selection analysis
sce <- runSoNNia(immLynx_example,
  chains = "TRB",
  background_file = bg_file)

# Get raw results instead of adding to object
sonia_results <- runSoNNia(immLynx_example,
  chains = "TRB",
  background_file = bg_file,
  return_object = FALSE)

## End(Not run)
```

---

runTCRdist

*Calculate TCR Distances on scRepertoire Data*

---

### Description

This function extracts TCR sequences from a SingleCellExperiment object with scRepertoire data and calculates pairwise TCR distances using tcrdist3.

**Usage**

```
runTCRdist(
  input,
  chains = "beta",
  organism = "human",
  compute_distances = TRUE,
  add_to_object = FALSE
)
```

**Arguments**

input	A SingleCellExperiment object containing scRepertoire TCR data.
chains	Character vector specifying chains: "alpha", "beta", or c("alpha", "beta"). Default is "beta".
organism	Organism: "human" or "mouse". Default is "human".
compute_distances	Logical. Whether to compute full distance matrix. Default TRUE.
add_to_object	Logical. If TRUE, attempts to add distance matrix to object (stored in metadata for SCE). Default is FALSE due to large matrix size.

**Value**

A list containing:

distances	Distance matrices (pw_alpha, pw_beta, pw_cdr3_a_aa, pw_cdr3_b_aa)
barcodes	Cell barcodes corresponding to matrix rows/columns
tcr_data	The formatted TCR data.frame used for analysis

If add\_to\_object=TRUE, returns the input object with distances stored.

**Examples**

```
data(immLynx_example)

# Calculate TCR distances for beta chain
dist_results <- runTCRdist(immLynx_example,
  chains = "beta")

# Access the distance matrix
dist_matrix <- dist_results$distances
barcodes <- dist_results$barcodes

# Calculate for both chains
dist_both <- runTCRdist(immLynx_example,
  chains = c("alpha", "beta"))

# Add distances directly to the object
sce <- runTCRdist(immLynx_example,
  chains = "beta",
```

```
add_to_object = TRUE)
```

---

```
summarizeTCRrepertoire
```

*Summarize TCR Repertoire Statistics*

---

## Description

Generates summary statistics for TCR repertoire data including diversity metrics, clonality measures, and sequence characteristics.

## Usage

```
summarizeTCRrepertoire(
  input,
  chains = c("TRB", "TRA", "both"),
  group.by = NULL,
  calculate_diversity = TRUE
)
```

## Arguments

input	A <code>SingleCellExperiment</code> object with <code>scRepertoire</code> data, or a <code>data.frame</code> from <code>extractTCRdata()</code> .
chains	Which chains to summarize: "TRB", "TRA", or "both". Default is "TRB".
group.by	Optional metadata column for grouping ( <code>SingleCellExperiment</code> objects only).
calculate_diversity	Logical. If TRUE, calculates diversity indices. Default is TRUE.

## Value

An object of class `TCR_summary` containing summary statistics for the TCR repertoire.

## Examples

```
# Summarize from a data.frame
tcr_data <- data.frame(
  barcode = paste0("cell_", 1:10),
  cdr3_aa = c("CASSLGTGELFF", "CASSIRSSYEQYF", "CASSLGTGELFF",
             "CASSYSTGELFF", "CASSIRSSYEQYF", "CASSLGTGELFF",
             "CASNQGLNEKLFF", "CASSYSTGELFF", "CASSLGTGELFF",
             "CASSIRSSYEQYF"),
  v = rep("TRBV7-2", 10),
  j = rep("TRBJ2-2", 10),
  chain = rep("TRB", 10),
  stringsAsFactors = FALSE
)
```

```
summary <- summarizeTCRrepertoire(tcr_data)
print(summary)
```

---

TCR\_summary-class      *S4 Class for TCR Repertoire Summary*

---

### Description

Formal S4 class to store summary statistics for a TCR repertoire, including diversity metrics, clonality measures, and sequence characteristics.

### Usage

```
## S4 method for signature 'TCR_summary'
show(object)
```

### Arguments

object                    A TCR\_summary object.

### Value

An S4 object of class TCR\_summary containing the summary statistics described in the slots. The show method prints a formatted summary to the console and returns invisible(NULL).

### Slots

total\_cells Integer. Total number of cells with TCR data.  
unique\_clonotypes Integer. Number of unique clonotypes.  
clonotype\_ratio Numeric. Ratio of unique clonotypes to total cells.  
diversity List of diversity indices (Shannon, Simpson, etc.), or NULL.  
top\_clones Data.frame of the top 10 most frequent clonotypes.  
cdr3\_length List with CDR3 length distribution statistics.  
gene\_usage List of V and J gene usage data.frames.  
chains Character. Chain(s) summarized.

---

tokenizeSequences	<i>Tokenize Amino Acid Sequences</i>
-------------------	--------------------------------------

---

### Description

Takes a vector of amino acid sequences and uses a Hugging Face tokenizer to convert them into numerical input IDs suitable for model input. The tokenizer should be obtained from [huggingModel](#), which manages the Python environment via basilisk.

### Usage

```
tokenizeSequences(  
  tokenizer,  
  aa_sequences,  
  padding = TRUE,  
  truncation = TRUE,  
  return_tensors = "pt"  
)
```

### Arguments

tokenizer	The tokenizer object returned by <a href="#">huggingModel</a> .
aa_sequences	A character vector of amino acid sequences (e.g., CDR3 sequences).
padding	A logical or string. If TRUE, pads sequences to the length of the longest sequence in the batch. Defaults to TRUE.
truncation	A logical. If TRUE, truncates sequences to the model's maximum input length. Defaults to TRUE.
return_tensors	A string specifying the format for the returned tensors. "pt" for PyTorch tensors, "tf" for TensorFlow. Defaults to "pt".

### Value

The tokenized output, typically a dictionary-like object containing 'input\_ids' and 'attention\_mask'.

### See Also

[huggingModel](#), [proteinEmbeddings](#), [runEmbeddings](#)

### Examples

```
sequences <- c("CASSLGTGELFF", "CASSIRSSYEQYF", "CASSYSTGELFF")  
  
# Initialize model and tokenizer  
hf_components <- huggingModel()  
  
# Tokenize CDR3 sequences  
tokenized <- tokenizeSequences(hf_components$tokenizer,
```

```

sequences)

# Tokenize without padding (variable-length output)
tokenized_nopad <- tokenizeSequences(
  hf_components$tokenizer,
  sequences,
  padding = FALSE)

# Pass tokenized output to proteinEmbeddings
embeddings <- proteinEmbeddings(hf_components$model,
  tokenized,
  pool = "mean",
  chunk_size = 32)

# Clean up
basilisk::basiliskStop(hf_components$proc)

```

---

 validateTCRdata

*Validate TCR Data Format*


---

### Description

Validates that TCR data is in the correct format for immLynx analysis functions. Checks for required columns, valid sequence formats, and gene nomenclature.

### Usage

```

validateTCRdata(
  tcr_data,
  check_genes = FALSE,
  check_sequences = TRUE,
  strict = FALSE
)

```

### Arguments

tcr_data	A data.frame containing TCR data
check_genes	Logical. If TRUE, validates gene names against IMGT nomenclature. Default is FALSE.
check_sequences	Logical. If TRUE, validates that CDR3 sequences contain only valid amino acids. Default is TRUE.
strict	Logical. If TRUE, stops with error on validation failure. If FALSE, returns validation report. Default is FALSE.

**Value**

If strict=FALSE, returns a list with:

**valid** Logical indicating overall validity

**errors** Character vector of error messages

**warnings** Character vector of warning messages

**summary** Summary statistics of the data

If strict=TRUE, returns TRUE invisibly on success or stops with error.

**Examples**

```
# Create example TCR data
tcr_data <- data.frame(
  barcode = paste0("cell_", 1:5),
  cdr3_aa = c("CASSLGTGELFF", "CASSIRSSYEQYF", "CASSYSTGELFF",
             "CASNQLNEKLFF", "CASLDRNEQFF"),
  v = paste0("TRBV", c("7-2", "12-3", "5-1", "28", "7-9")),
  j = paste0("TRBJ", c("2-2", "1-1", "2-7", "1-5", "2-1")),
  chain = rep("TRB", 5),
  stringsAsFactors = FALSE
)
report <- validateTCRdata(tcr_data, strict = FALSE)
report$valid
report$summary
```

# Index

## \* datasets

- [immLynx\\_example](#), [6](#)
- [convertToTcrdist](#), [2](#)
- [extractTCRdata](#), [3](#)
- [generateOLGA](#), [5](#)
- [huggingModel](#), [5](#), [8](#), [21](#)
- [immLynx\\_example](#), [6](#)
- [proteinEmbeddings](#), [6](#), [7](#), [21](#)
- [runClustTCR](#), [9](#)
- [runEmbeddings](#), [6–8](#), [10](#), [21](#)
- [runHLAassociation](#), [12](#)
- [runMetaclonotypist](#), [13](#)
- [runOLGA](#), [15](#)
- [runSoNNia](#), [16](#)
- [runTCRdist](#), [17](#)
- [show](#), [TCR\\_summary-method](#)  
([TCR\\_summary-class](#)), [20](#)
- [summarizeTCRrepertoire](#), [19](#)
- [TCR\\_summary](#), [19](#)
- [TCR\\_summary-class](#), [20](#)
- [tokenizeSequences](#), [6](#), [8](#), [21](#)
- [validateTCRdata](#), [22](#)