Package: SpliceImpactR (via r-universe)

May 30, 2026

Type Package

Title An R package to identify functional impacts due to alternative RNA processing events

Version 1.1.0

Description Works by taking in processed data from the HIT Index and/or rMATS and identifying how differentially used alternative RNA processing events lead to changes in protein function through various means. Primarily this is done through protein similarity, functional protein domain analysis, and domain-domain interaction changes. Notably, we both identify alterantive RNA processing event 'swaps' across condition and are able to perform holistic analyses regarding the impact of different RNA processing events.

License GPL-3

Encoding UTF-8

Imports data.table, BiocFileCache, BiocParallel, Biostrings, GenomicRanges, SummarizedExperiment, biomaRt, IRanges, PFAM.db, dplyr, ggplot2, ggpubr, patchwork, pwalign, rtracklayer, scales, stats, tidyr, tools, utils, magrittr, methods, S4Vectors

RoxygenNote 7.3.3

biocViews AlternativeSplicing, DifferentialSplicing, StatisticalMethod, Alignment

Suggests devtools, testthat (>= 3.0.0), knitr, rmarkdown, cowplot, stringr, readr, tibble, BiocStyle, clusterProfiler, AnnotationDbi, msigdbr, org.Hs.eg.db, org.Mm.eg.db

VignetteBuilder knitr

Config/testthat/edition 3

BugReports https://github.com/fiszbein-lab/SpliceImpactR/issues

Config/pak/sysreqs cmake make libbz2-dev libicu-dev liblzma-dev libpng-dev libxml2-dev libssl-dev xz-utils zlib1g-dev

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

Date/Publication 2026-04-28 13:06:51 UTC

RemoteUrl https://github.com/bioc/SpliceImpactR

RemoteRef HEAD

RemoteSha 64b59abd29e1fab8340fd18f9bd4a9090606507f

Contents

`add_splice_part` *Add one part to an existing SpliceImpactResult*

Description

Add one part to an existing SpliceImpactResult

Usage

```
add_splice_part(
  obj,
  data = NULL,
  res = NULL,
  res_di = NULL,
  matched = NULL,
  sample_frame = NULL,
  hits_final = NULL
)
```

Arguments

`obj`	'SpliceImpactResult'
`data`	Optional raw sample-level table.
`res`	Optional differential result table.
`res_di`	Optional threshold-filtered differential table.
`matched`	Optional annotation-matched table.

sample_frame	Optional sample manifest table.
hits_final	Optional paired/final hits table.

Value

Updated ‘SpliceImpactResult‘

Examples

```
obj <- as_splice_impact_result()
res <- data.table::data.table(
  event_id = c("E1", "E1"),
  form = c("inc", "exc"),
  inc = c("100-110", "120-130"),
  exc = c("120-130", "100-110"),
  chr = c("chr1", "chr1"),
  strand = c("+", "+"),
  gene_id = c("ENSG000001", "ENSG000001"),
  padj = c(0.01, 0.01),
  delta_psi = c(0.25, -0.25)
)
obj <- add_splice_part(obj, res = res)
print(as_dt_from_s4(obj, "di_events"))
```

---

as_dt_from_s4 *Convert S4 slots back to data.table*

Description

Convert S4 slots back to data.table

Usage

```
as_dt_from_s4(
  x,
  slot = c("raw_events", "di_events", "res_di", "matched", "sample_frame",
    "hits_sequences", "paired_hits"),
  keep_internal_keys = FALSE
)
```

Arguments

x	‘SpliceImpactResult‘
slot	One of ‘raw_events‘, ‘di_events‘, ‘res_di‘, ‘matched‘, ‘sample_frame‘, ‘paired_hits‘. For backward compatibility, ‘"hits_sequences"‘ is treated as ‘"matched"‘.
keep_internal_keys	Keep internal key columns (‘raw_key‘, ‘di_key‘, ‘pair_key‘).

Value

'data.table'

Examples

```
res <- data.table::data.table(
  event_id = c("E1", "E1"),
  form = c("inc", "exc"),
  inc = c("100-110", "120-130"),
  exc = c("120-130", "100-110"),
  chr = c("chr1", "chr1"),
  strand = c("+", "+"),
  gene_id = c("ENSG000001", "ENSG000001"),
  padj = c(0.01, 0.01),
  delta_psi = c(0.20, -0.20)
)
obj <- as_splice_impact_result(res = res)
print(as_dt_from_s4(obj, "di_events"))
```

---

as_splice_impact_result

Build S4 SpliceImpact container

Description

Accepts any subset of 'data', 'res', and 'hits_final'. Missing pieces are stored as empty valid slots and can be added later with [add_splice_part()].

Usage

```
as_splice_impact_result(
  data = NULL,
  res = NULL,
  res_di = NULL,
  matched = NULL,
  sample_frame = NULL,
  hits_final = NULL,
  metadata = list()
)
```

Arguments

data	Optional sample-level input table.
res	Optional differential inclusion result table.
res_di	Optional threshold-filtered differential inclusion table.
matched	Optional annotation-matched DI table.
sample_frame	Optional sample manifest table.

hits_final	Optional paired/final hit table.
metadata	Optional list.

Value

'SpliceImpactResult'

Examples

```
raw <- data.table::data.table(
  event_id = c("E1", "E1"),
  form = c("inc", "exc"),
  sample = c("S1", "S1"),
  chr = c("chr1", "chr1"),
  strand = c("+", "+"),
  inc = c("100-110", "120-130"),
  exc = c("120-130", "100-110"),
  psi = c(0.70, 0.30),
  inclusion_reads = c(70, 30),
  exclusion_reads = c(30, 70)
)
res <- data.table::data.table(
  event_id = c("E1", "E1"),
  form = c("inc", "exc"),
  inc = c("100-110", "120-130"),
  exc = c("120-130", "100-110"),
  chr = c("chr1", "chr1"),
  strand = c("+", "+"),
  gene_id = c("ENSG000001", "ENSG000001"),
  padj = c(0.01, 0.01),
  delta_psi = c(0.20, -0.20)
)
hits <- data.table::data.table(
  event_id = "E1",
  event_type = "SE",
  gene_id = "ENSG000001",
  chr = "chr1",
  strand = "+",
  transcript_id_control = "TX1",
  transcript_id_case = "TX2",
  inc_control = "100-110",
  inc_case = "100-115",
  exc_control = "120-130",
  exc_case = "121-130",
  n_ppi = 1L,
  diff_n = 1L
)
obj <- as_splice_impact_result(data = raw, res = res, hits_final = hits)
obj
```

attach_sequences *Attach transcript and protein sequences to an event or annotation table*

Description

Merges sequence data (transcript and protein sequences) onto an input table of splicing events or transcript annotations, by matching on `transcript_id`. When multiple sequences share the same `transcript_id`, the function keeps the entry with a non-missing `protein_id` and the longest `protein_seq`.

Usage

```
attach_sequences(x, sequences, return_class = c("auto", "data.table", "S4"))
```

Arguments

`x`	A data.frame or data.table containing a `transcript_id` column.
`sequences`	A data.frame or data.table with at least the columns: `transcript_id`, `protein_id`, `transcript_seq`, and `protein_seq`.
`return_class`	Character. Output mode: ‘"data.table"‘, ‘"S4"‘, or ‘"auto"‘ (default). In ‘auto‘, S4 input returns updated S4 output.

Details

The join is left-sided: all rows from `x` are preserved. Duplicate `transcript_id`s in `sequences` are resolved internally based on protein presence and sequence length.

Value

A data.table with the same rows as `x` and appended sequence columns (`transcript_seq`, `protein_seq`, etc.).

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
annots <- load_example_data("annotation_df")$annotation_df
matched <- get_matched_events_chunked(res, annots$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annots$sequences)
print(x_seq)
```

`compare_hit_index` *Compare HIT index values between conditions*

Description

Computes per-event differences in the **HIT index** (Hybrid Intron Tolerance) between control and test conditions, performs t test, adjusts for multiple testing, and produces summary visualizations.

Usage

```
compare_hit_index(
  sample_df,
  condition_map = c(control = "control", test = "case"),
  minimum_proportion = 0.5,
  top_n_heatmap = 5000,
  sig_delta = 0.2,
  fdr_thresh = 0.05
)
```

Arguments

`sample_df`	‘data.frame‘ or ‘data.table‘ containing sample metadata with columns: ‘path‘, ‘sample_name‘, and ‘condition‘.
`condition_map`	Named character vector mapping experimental groups, e.g. ‘c(control = "control", test = "case")‘.
`minimum_proportion`	Minimum proportion of samples per group that must have valid HIT values for a test to be performed (default = 0.5).
`top_n_heatmap`	Integer; number of events to display in the heatmap ordered by absolute deltaHIT (default = 5000).
`sig_delta`	Absolute deltaHIT threshold for volcano highlighting (default = 0.2).
`fdr_thresh`	FDR threshold for volcano significance lines (default = 0.05).

Details

For each exon-level HIT index event:

- Computes mean HIT per group (‘control‘, ‘test‘)
- Performs a t test if both groups have >50
- Adjusts p-values via Benjamini-Hochberg FDR
- Computes signed (‘diff_HIT‘) and absolute (‘delta_HIT‘) changes

The function returns both the full per-event results table and a combined 2x2 patchwork plot containing:

1. Control vs Test mean HIT scatter
2. Top |deltaHIT| heatmap (control/test columns)
3. Volcano plot of |deltaHIT| vs -log10(FDR)
4. Density of deltaHIT distribution

Value

A named list with:

'results' Data.table of per-event statistics (means, p, FDR, deltaHIT)

'plot' Patchwork object with 4-panel summary visualization

See Also

[.getHITindex()]

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_compare <- compare_hit_index(sample_frame, condition_map = c(control = "control", test = "case"))
print(hit_compare)
```

`compare_sequence_frame`

Compare frame states after sequence alignment

Description

Wrapper function that performs sequence alignment (via [compare_sequences_alignment()]) and frame-shift analysis (via [compare_frames()]) for a complete set of inclusion/exclusion transcript pairs. It then summarizes each event by a high-level classification label.

Usage

```
compare_sequence_frame(
  complete_hits,
  ann,
  return_class = c("auto", "data.table", "S4")
)
```

Arguments

complete_hits	‘data.frame‘, ‘data.table‘, or ‘SpliceImpactResult‘ containing complete event information for inclusion/exclusion transcript pairs, typically from [get_pairs()] or similar.
ann	Annotation object (output of [get_annotation()]) used for both sequence alignment and coding index construction.
return_class	Character. Output mode: ‘"data.table"‘, ‘"S4"‘, or ‘"auto"‘ (default). In ‘auto‘, S4 input returns updated S4 output.

Details

‘compare_sequence_frame()‘ is a convenience function that integrates sequence and frame comparison stages in one call, producing an annotated table suitable for downstream summarization or visualization.

The summary label ‘summary_classification‘ follows this precedence: 1. ‘"Match"‘ - identical protein sequences. 2. ‘"FrameShift"‘ - frame disrupted. 3. ‘"Rescue"‘ - frame restored downstream. 4. Otherwise, inherited from ‘pc_class‘.

Value

A ‘data.table‘ (or updated ‘SpliceImpactResult‘ when ‘return_class‘ resolves to S4) containing all columns from ‘complete_hits‘, plus:

frame_call Result from [compare_frames()].

rescue Rescue classification.

summary_classification One of ‘"FrameShift"‘, ‘"Rescue"‘, ‘"Match"‘, or the original ‘pc_class‘.

See Also

[compare_frames()], [compare_sequences_alignment()]

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
annots <- load_example_data("annotation_df")$annotation_df
matched <- get_matched_events_chunked(res, annots$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annots$sequences)
pairs <- get_pairs(x_seq, source="multi")
seq_compare <-compare_sequence_frame(pairs, annots$annotations)
print(seq_compare)
```

compare_sequences_alignment *Compare isoform nucleotide and protein sequences by pairwise alignment*

Description

Performs transcript- and protein-level global alignments between included and excluded isoform sequences to quantify sequence similarity, coding differences, and exon coverage differences.

Usage

```
compare_sequences_alignment(
  hits,
  annotations,
  include_sequences = FALSE,
  verbose = TRUE
)
```

Arguments

hits	A data.table or data.frame containing isoform pairs. Must include columns: transcript_seq_case, transcript_seq_control, protein_seq_case, and protein_seq_control. From prior analysis.
annotations	output from get_annotations (annotations)
include_sequences	Logical; if TRUE, retains raw sequences in the output. Defaults to FALSE.
verbose	Logical; if TRUE, prints processing messages.

Details

This function wraps internal helpers for alignment (.align_dna(), .align_aa()), exon length summarization (.sum_exon_lengths()), and protein-coding classification (.pc_class()).

Value

A data.table with added columns describing sequence length, coding length, and alignment similarity metrics, including:

- pc_class: protein-coding status ("protein_coding", "onePC", "noPC")
- prot_len_case/control: protein sequence lengths
- tx_len_case/control: transcript sequence lengths
- exon_cds_len_*, exon_len_*: summed exon/CDS lengths
- dna_pid, dna_score, prot_pid, prot_score: alignment metrics

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
annots <- load_example_data("annotation_df")$annotation_df
matched <- get_matched_events_chunked(res, annots$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annots$sequences)
pairs <- get_pairs(x_seq, source="multi")
aligned <- compare_sequences_alignment(pairs, annots$annotations)
print(aligned)
```

`compare_transcript_pairs`

Compare user-selected transcript pairs

Description

Builds a matched-like table for pairs of transcripts by extracting all coding exon coordinates from annotations.

Usage

```
compare_transcript_pairs(transcript_pairs, annotations)
```

Arguments

`transcript_pairs`	data.frame with columns ‘transcript1‘, ‘transcript2‘
`annotations`	flattened GTF-style data.frame or data.table (from get_annotation)

Value

data.table mimicking ‘matched‘ structure, ready for downstream comparison.

Examples

```
annotation_df <- load_example_data("annotation_df")$annotation_df
pairs <- data.frame(
    transcript1 = c("ENST00000337907", "ENST00000426559"),
    transcript2 = c("ENST00000400908", "ENST00000399728")
)
matched <- compare_transcript_pairs(pairs, annotation_df$annotations)
print(matched)
```

enrich_by_db *Run domain enrichment by database*

Description

Convenience wrapper for [enrich_domains_hypergeo()] that runs the enrichment test separately for each database (e.g. ‘"Pfam"‘, ‘"SMART"‘) and combines the results.

Usage

```
enrich_by_db(hits, background, dbs, ...)
```

Arguments

hits, background	See [enrich_domains_hypergeo()].
dbs	Character vector of database prefixes to test.
...	Additional arguments passed to [enrich_domains_hypergeo()].

Value

Combined ‘data.table‘ with an added ‘database‘ column.

See Also

[enrich_by_event()]

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
annotation_df <- load_example_data("annotation_df")$annotation_df
matched <- get_matched_events_chunked(res, annotation_df$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annotation_df$sequences)
pairs <- get_pairs(x_seq, source="multi")
seq_compare <-compare_sequence_frame(pairs, annotation_df$annotations)
annotation_df <- get_annotation(load = 'test')
interpro_features <- get_protein_features(c("interpro"), annotation_df$annotations, timeout = 600, test = TRUE)
protein_feature_total <- get_comprehensive_annotations(list(interpro_features))

exon_features <- get_exon_features(annotation_df$annotations, protein_feature_total)

hits_domain <- get_domains(seq_compare, exon_features)

bg <- get_background(source = "hit_index",
                     input = sample_frame,
                     annotations = annotation_df$annotations,
```

```
                    protein_features = protein_feature_total)

enriched_domains <- enrich_by_db(hits_domain, bg, dbs = 'interpro')
print(enriched_domains)
```

`enrich_by_event` *Run domain enrichment by event type*

Description

Convenience wrapper for [enrich_domains_hypergeo()] that runs the enrichment test separately for each event type and combines results.

Usage

```
enrich_by_event(hits, background, events, ...)
```

Arguments

`hits`, `background`	See [enrich_domains_hypergeo()].
`events`	Character vector of event types to analyze.
`...`	Additional arguments passed to [enrich_domains_hypergeo()].

Value

Combined ‘data.table‘ with an added ‘event_type‘ column.

See Also

[enrich_by_db()]

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
annotation_df <- load_example_data("annotation_df")$annotation_df
matched <- get_matched_events_chunked(res, annotation_df$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annotation_df$sequences)
pairs <- get_pairs(x_seq, source="multi")
seq_compare <-compare_sequence_frame(pairs, annotation_df$annotations)
annotation_df <- get_annotation(load = 'test')
interpro_features <- get_protein_features(c("interpro"), annotation_df$annotations, timeout = 600, test = TRUE)
protein_feature_total <- get_comprehensive_annotations(list(interpro_features))
```

```
exon_features <- get_exon_features(annotation_df$annotations, protein_feature_total)

hits_domain <- get_domains(seq_compare, exon_features)

bg <- get_background(source = "hit_index",
                     input = sample_frame,
                     annotations = annotation_df$annotations,
                     protein_features = protein_feature_total)

enriched_domains <- enrich_by_event(hits_domain, bg, events = 'AFE', db_filter = 'interpro')
print(enriched_domains)
```

`enrich_domains_hypergeo` *Domain-level enrichment via hypergeometric test*

Description

Tests whether particular protein domains are overrepresented among inclusion/exclusion transcript pairs (foreground) relative to a matched background set, using the hypergeometric test.

Usage

```
enrich_domains_hypergeo(
  hits,
  background,
  domain_col_fg = "either_domains_list",
  domain_col_bg = "total_sd_domains",
  event_col = "event_type",
  event_filter = NULL,
  db_filter = NULL,
  min_fg_count = 2,
  delim = "[,;|[:space:]]+"
)
```

Arguments

`hits`	‘data.frame‘ or ‘data.table‘ containing the foreground transcript pairs (typically the significant inclusion/exclusion events). Must include a list column of domain IDs (default: ‘"either_domains_list"‘).
`background`	‘data.frame‘ or ‘data.table‘ representing the matched background pairs. Must include a list column of domain IDs (default: ‘"total_sd_domains"‘).
`domain_col_fg`	Name of the domain list column in ‘hits‘.
`domain_col_bg`	Name of the domain list column in ‘background‘.

event_col	Name of the column giving event type (default: ‘"event_type"‘). Set ‘NULL‘ to skip event filtering.
event_filter	Character vector of event types to include (e.g. ‘"A5SS"‘, ‘"A3SS"‘). If ‘NULL‘, all events are used.
db_filter	Character vector of database prefixes to retain (e.g. ‘"Pfam"‘, ‘"SMART"‘). If ‘NULL‘, all domains are used.
min_fg_count	Minimum number of foreground hits required to test a domain (default ‘2‘).
delim	Regular expression describing the delimiters in string list columns (default ‘[,;|[:space:]]+‘).

Details

Each domain identifier is counted once per transcript pair based on its presence in a list column (e.g. ‘either_domains_list‘). The probability of observing at least ‘k‘ such pairs is computed under the hypergeometric distribution

$$P(X \geq k), \quad X \sim \mathrm{Hypergeom}(M, B - M, K)$$

where:

- K = number of foreground pairs,
- B = number of background pairs,
- M = background count of pairs containing the domain,
- k = foreground count of pairs containing the domain.

P-values are Benjamini-Hochberg adjusted (‘padj‘).

Optionally, analyses can be restricted by event type or database prefix (e.g. ‘"Pfam"‘, ‘"SMART"‘) and domains with fewer than ‘min_fg_count‘ foreground occurrences are skipped.

Value

A ‘data.table‘ with one row per domain, including:

‘domain_id‘ Domain identifier (database prefix removed).

‘db‘ Database prefix (e.g. ‘"Pfam"‘).

‘k‘, ‘K‘ Foreground domain count and total foreground pairs.

‘M‘, ‘B‘ Background domain count and total background pairs.

‘fg_prop‘, ‘bg_prop‘ Proportion of pairs with the domain.

‘OR‘ Odds ratio (Haldane-Anscombe corrected).

‘pval‘, ‘padj‘ Raw and BH-adjusted p-values.

‘events‘ Event IDs contributing to the domain count.

See Also

* [enrich_by_event()] run per event type * [enrich_by_db()] run per database * [add_domain_columns()] attach domain lists to hits

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
annotation_df <- load_example_data("annotation_df")$annotation_df
matched <- get_matched_events_chunked(res, annotation_df$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annotation_df$sequences)
pairs <- get_pairs(x_seq, source="multi")
seq_compare <-compare_sequence_frame(pairs, annotation_df$annotations)
annotation_df <- get_annotation(load = 'test')
interpro_features <- get_protein_features(c("interpro"), annotation_df$annotations, timeout = 600, test = TRUE)
protein_feature_total <- get_comprehensive_annotations(list(interpro_features))

exon_features <- get_exon_features(annotation_df$annotations, protein_feature_total)

hits_domain <- get_domains(seq_compare, exon_features)

bg <- get_background(source = "hit_index",
                     input = sample_frame,
                     annotations = annotation_df$annotations,
                     protein_features = protein_feature_total)

enriched_domains <- enrich_domains_hypergeo(hits_domain, bg, db_filter = 'interpro')
print(enriched_domains)
```

---

filter_spliceimpact_hits

Filter a SpliceImpactResult by arbitrary paired-hit columns

Description

Filters ‘paired_hits‘ using one or more logical expressions evaluated in the paired-hit table, then synchronizes all event-linked slots (‘segments‘, ‘res_di‘, ‘di_events‘, ‘matched‘, ‘raw_events‘).

Usage

```
filter_spliceimpact_hits(obj, ..., keep_sample_frame = TRUE)
```

Arguments

obj	A [SpliceImpactResult].
...	Logical filter expressions evaluated in paired-hit context (e.g., ‘event_id == "A3SS:44"‘, ‘n_ppi > 0‘, ‘frame_call == "Match"‘). Multiple expressions are combined with ‘&‘.
keep_sample_frame	Logical; keep ‘sample_frame‘ unchanged (default ‘TRUE‘).

Value

Filtered [SpliceImpactResult].

Examples

```
hits <- data.table::data.table(
  event_id = c("E1", "E2"),
  event_type = c("SE", "A3SS"),
  gene_id = c("ENSG000001", "ENSG000002"),
  chr = c("chr1", "chr2"),
  strand = c("+", "-"),
  transcript_id_control = c("TX1", "TX3"),
  transcript_id_case = c("TX2", "TX4"),
  inc_control = c("100-110", "200-210"),
  inc_case = c("100-115", "205-215"),
  exc_control = c("120-130", "220-230"),
  exc_case = c("121-130", "225-235"),
  n_ppi = c(1L, 0L),
  diff_n = c(1L, 0L),
  frame_call = c("Match", "Frameshift")
)
obj <- as_splice_impact_result(hits_final = hits)
obj_keep <- filter_spliceimpact_hits(obj, n_ppi > 0L)
print(as_dt_from_s4(obj_keep, "paired_hits"))
```

---

get_annotation *Load and cache GENCODE annotations, sequences, and hybrid exon annotations*

Description

This function loads GENCODE gene models (GTF), processes exon annotations, extracts transcript and protein sequences, identifies hybrid exons, and optionally caches the processed objects for future fast access.

Usage

```
get_annotation(
  load = c("link", "path", "cached", "test"),
  base_dir = NULL,
  species = c("human", "mouse"),
  release = 45,
  gtf_path = NULL,
  transcript_path = NULL,
  translation_path = NULL,
  filter_tsl = c("1", "2", "3")
)
```

Arguments

load	Character string specifying load mode: one of ‘"link"‘, ‘"path"‘, ‘"cached"‘, ‘"test"‘.
base_dir	Optional cache root. If ‘NULL‘ (default), uses a persistent package cache under ‘tools::R_user_dir("SpliceImpactR", "cache")‘. A package-specific ‘BiocFileCache‘ is created under this root.
species	Species label used in filenames (default ‘"human"‘).
release	GENCODE release version (default ‘45‘).
gtf_path	Path to a GTF file when ‘load = "path"‘.
transcript_path	Path to transcript FASTA (.fa/.fa.gz) when ‘load = "path"‘.
translation_path	Path to protein FASTA (.fa/.fa.gz) when ‘load = "path"‘.
filter_tsl	Transcript support levels to retain (default ‘c("1","2","3")‘). Transcripts outside this set are dropped unless the row is a gene record.

Details

Four loading modes are supported:

‘test‘ Load small internal test data shipped with the package.

‘cached‘ Load previously processed objects from package ‘BiocFileCache‘.

‘path‘ Read local GTF and FASTA files, process, then cache processed objects in package ‘BiocFileCache‘.

‘link‘ Download GENCODE files from URLs, process, then cache processed objects in package ‘BiocFileCache‘. Optional ‘gtf_path‘, ‘transcript_path‘, and ‘translation_path‘ are only used as overrides when they are existing local files or valid URLs; otherwise downloaded GENCODE assets are used.

Processed objects are cached in package ‘BiocFileCache‘ entries:

```
annotation/{species}/v{release}/tsl-{...}/annotations.rds
annotation/{species}/v{release}/tsl-{...}/sequences.rds
annotation/{species}/v{release}/tsl-{...}/hybrids.rds
```

Value

A list with:

‘annotations‘ Processed long-format GTF as ‘data.table‘

‘sequences‘ List with elements ‘transcripts‘ and ‘proteins‘ (or ‘NULL‘ if not loaded)

‘hybrids‘ Hybrid exon annotation list

Examples

```
# Load bundled test data
ann <- load_example_data("annotation_df")$annotation_df
print(ann)

# Load from local files and cache processed objects
# ann <- get_annotation(
#   load = "path",
#   gtf_path = "/downloaded_gtf_directory/gencode.v45.annotation.gtf.gz",
#   transcript_path = "/downloaded_gtf_directory/gencode.v45.pc_transcripts.fa.gz",
#   translation_path = "/downloaded_gtf_directory/gencode.v45.pc_translations.fa.gz"
# )

# Download files, process, and cache to a custom cache root
# ann <- get_annotation(
#   load = "link",
#   base_dir = "/project/annotation_cache/"
# )

# Load from cached RDS (fast)
# ann <- get_annotation(
#   load = "cached",
#   base_dir = "/project/annotation_cache/"
# )
```

---

get_background *Build a transcript-pair background with domain and length annotations*

Description

Constructs a background dataset of transcript pairs suitable for domain-level or exon-level enrichment analyses. Depending on the `source` parameter, the function can derive the background from HIT index results, annotated transcripts, or a user-provided list of transcript IDs.

Usage

```
get_background(
  source = c("hit_index", "annotated", "user-given"),
  input,
  annotations,
  protein_features,
  keep_annotated_first_last = TRUE,
  minOverlap = 0.8,
  BPPARAM = BiocParallel::bpparam()
)
```

Arguments

source	Character string specifying the source of the background. One of: • "hit_index" use HIT index output directories. • "annotated" use all transcripts from the annotation. • "user-given" use a user-supplied list of transcript IDs.
input	Source-specific input: • For "hit_index": a data.frame of paths with a path column. • For "annotated": ignored. • For "user-given": a character vector or data.frame containing transcript_id.
annotations	A data.table annotations from get_annotations
protein_features	A data.frame of protein domain or motif features (e.g. InterPro, Pfam) with at least ensembl_transcript_id, ensembl_peptide_id, database, and name columns from get_comprehensive_annotations
keep_annotated_first_last	Logical; passed to read_background to control filtering of annotated first/last exons. Defaults to TRUE.
minOverlap	Numeric (0-1); minimum fraction overlap required when matching exons between HIT index data and annotations. Defaults to 0.8.
BPPARAM	A [BiocParallel::BiocParallelParam-class] object controlling parallel execution in domain background calculations. Defaults to [BiocParallel::bpparam()].

Value

A data.table in which each row represents a transcript pair annotated with gene ID, CDS/exon length differences, and protein domain differences. Columns include: gene_id, transcript_id_1, transcript_id_2, prot_aa_1, prot_aa_2, domains_1, domains_2, and related summary metrics.

Examples

```
annots <- load_example_data("annotation_df")$annotation_df
interpro_features <- get_protein_features(c("interpro"), annots$annotations, timeout = 600, test = TRUE)
protein_feature_total <- get_comprehensive_annotations(list(interpro_features))

# Build background from HIT index paths
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
bg <- get_background(source = "hit_index",
                     input = sample_frame,
                     annotations = annots$annotations,
                     protein_features = protein_feature_total)

# Or from user-supplied transcript IDs
tx_ids <- c("ENST00000466994","ENST00000484435")
bg_user <- get_background(source = "user-given",
                          input = tx_ids,
                          annotations = annots$annotations,
```

```
                        protein_features = protein_feature_total)
```

`get_comprehensive_annotations` *Combine multiple sources of protein feature annotations*

Description

Aggregates all available feature tables (e.g., biomaRt + manual) into a single unified long-format annotation table.

Usage

```
get_comprehensive_annotations(
  protein_feature_list,
  load_path_list = NULL,
  save_path = NULL
)
```

Arguments

`protein_feature_list`	List of `data.table` objects, typically from [get_protein_features()] or [get_manual_features()].
`load_path_list`	Optional vector of file paths to load each feature source from disk (instead of providing in memory).
`save_path`	Optional path to cache the combined annotations.

Value

A combined `data.table` containing all unique feature rows.

Examples

```
annotation_df <- load_example_data("annotation_df")$annotation_df
user_df <- data.frame(
  ensembl_transcript_id = c(
    "ENST00000511072","ENST00000374900","ENST00000373020","ENST00000456328",
    "ENST00000367770","ENST00000331789","ENST00000335137","ENST00000361567",
    NA,                    "ENST00000380152"
  ),
  ensembl_peptide_id = c(
    "ENSP00000426975", NA,                    "ENSP00000362048","ENSP00000407743",
    "ENSP00000356802","ENSP00000326734", NA,                   "ENSP00000354587",
    "ENSP00000364035", NA
  ),
  name = c(
```

```
    "Low complexity","Transmembrane helix","Coiled-coil","Signal peptide",
    "Transmembrane helix","Low complexity","Coiled-coil","Transmembrane helix",
    "Signal peptide","Low complexity"
  ),
  start = c(80L, 201L, 35L, 1L, 410L, 150L, 220L, 30L, 1L, 300L),
  stop  = c(120L,223L, 80L, 20L, 430L, 190L, 260L, 55L, 24L, 360L),
 database  = c("seg","tmhmm","ncoils","signalp","tmhmm","seg","ncoils","tmhmm","signalp", NA),
 alt_name  = c(NA,"TMhelix",NA,"SignalP-noTM", "TMhelix", NA, NA, "TMhelix", "SignalP-TAT", NA),
  feature_id = c(NA, NA, NA, NA, NA, NA, NA, NA, NA, NA)
)
user_features <- get_manual_features(user_df, annotation_df$annotations)
interpro_features <- get_protein_features(c("interpro"), annotation_df$annotations, timeout = 600, test = TRUE)
protein_feature_total <- get_comprehensive_annotations(list(user_features, interpro_features))
print(protein_feature_total)
```

`get_di_gene_enrichment` *Extract Differential-Inclusion Genes for Enrichment*

Description

Select genes whose splicing passes significance and effect-size thresholds.

Usage

```
get_di_gene_enrichment(hits, padj_threshold, delta_psi_threshold)
```

Arguments

`hits`	Data frame output from differential inclusion testing.
`padj_threshold`	FDR cutoff (default '0.05').
`delta_psi_threshold`	Absolute deltaPSI threshold (default '0.1').

Value

Character vector of gene IDs.

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)

annotation_df <- load_example_data("annotation_df")$annotation_df
interpro_features <- get_protein_features(c("interpro"), annotation_df$annotations, timeout = 600, test = TRUE)
protein_feature_total <- get_comprehensive_annotations(list(interpro_features))
```

```
exon_features <- get_exon_features(annotation_df$annotations, protein_feature_total)

matched <- get_matched_events_chunked(res, annotation_df$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annotation_df$sequences)
pairs <- get_pairs(x_seq, source="multi")
seq_compare <-compare_sequence_frame(pairs, annotation_df$annotations)
hits_domain <- get_domains(seq_compare, exon_features)

bg <- get_background(source = "hit_index",
                     input = sample_frame,
                     annotations = annotation_df$annotations,
                     protein_features = protein_feature_total)
enrichment <- get_enrichment(get_di_gene_enrichment(res, .05, .1), bg$gene_id, species = 'human', 'ensembl', 'MSig
print(enrichment)
```

`get_differential_inclusion`

Differential Inclusion Analysis from Hit Index Tables

Description

Performs per-site differential inclusion testing from a hit-index or junction-form table. Each site is modeled with a quasi-binomial GLM (‘psi_adj ~ condition‘) to estimate deltaPSI and significance, optionally using parallel processing.

Usage

```
get_differential_inclusion(
  DT,
  min_total_reads = 10L,
  minimum_proportion_containing_event = 0.5,
  terminal_fill = "event_max",
  cooks_cutoff = "Inf",
  adjust_method = "fdr",
  verbose = TRUE,
  parallel_glm = TRUE,
  chunk_size_glm = 1000,
  BPPARAM = BiocParallel::SerialParam(),
  return_class = c("auto", "data.table", "S4")
)
```

Arguments

`DT`	A ‘data.frame‘, ‘data.table‘, or ‘SpliceImpactResult‘ containing at least the columns ‘event_type‘, ‘gene_id‘, ‘chr‘, ‘inc‘, ‘exclusion_reads‘, ‘inclusion_reads‘, ‘condition‘, and ‘sample‘.

`min_total_reads`
: Integer. Minimum total reads per site/sample required for inclusion (default ‘10‘).

`minimum_proportion_containing_event`
: Numeric in ‘[0,1]‘. Minimum fraction of samples per condition that must contain the event (default ‘0.5‘).

`terminal_fill`
: Character or numeric. Strategy for completing AFE/ALE events that are missing in a given sample. Choose one of:

  "none" Do not add missing terminal sites.

  "gene_max" Fill missing sites with zero counts and set ‘total‘ to the maximum observed within each ‘gene_id‘/‘sample‘/‘condition‘ group.

  "event_max" Fill missing sites with zero counts and set ‘total‘ to the maximum observed within each ‘event_id‘/‘sample‘/‘condition‘ group.

  "zero" Fill missing sites with zero counts and ‘total = 0‘.

  Alternatively, supply a single numeric value to use as the ‘total‘ for all filled rows. Defaults to ‘"gene_max"‘.

`cooks_cutoff`
: Character or numeric. Cook’s distance cutoff: ‘"4/n"‘, ‘"Inf"‘, ‘"none"‘, or a numeric value.

`adjust_method`
: Character. Multiple-testing correction method passed to [stats::p.adjust()] (default ‘"fdr"‘).

`verbose`
: Logical. Print progress messages (default ‘TRUE‘).

`parallel_glm`
: Logical. Use parallel fitting via [fit_sites_parallel()] (default ‘TRUE‘).

`chunk_size_glm`
: Integer. Number of sites per parallel chunk (default ‘1000‘).

`BPPARAM`
: A [BiocParallel::BiocParallelParam-class] object used when ‘parallel_glm = TRUE‘. Default is [BiocParallel::SerialParam()].

`return_class`
: Character. Output mode: ‘"data.table"‘, ‘"S4"‘, or ‘"auto"‘ (default). In ‘auto‘, S4 input returns an updated S4 object; otherwise a ‘data.table‘ is returned.

Details

The function filters out low-coverage and low-presence events, optionally fills AFE/ALE sites with zero counts where necessary, and then applies site-level GLMs. Parallelization uses [BiocParallel] back-ends for reproducibility across platforms. To run in parallel, supply ‘BPPARAM‘ (for example [BiocParallel::MulticoreParam()] on Linux/macOS or [BiocParallel::SnowParam()] on Windows).

Value

If ‘return_class‘ resolves to ‘"data.table"‘, a ‘data.table‘ with one row per site containing:

- Metadata columns (‘site_id‘, ‘event_type‘, ‘event_id‘, ‘gene_id‘, ...)
- Sample counts (‘n_samples‘, ‘n_control‘, ‘n_case‘)
- Mean PSI per group (‘mean_psi_ctrl‘, ‘mean_psi_case‘)
- deltaPSI (‘delta_psi‘)
- Raw and adjusted p-values (‘p.value‘, ‘padj‘)
- Maximum Cook’s distance (‘cooks_max‘)

See Also

[fit_sites_parallel()], [cutoff_num()], [stats::glm()], [BiocParallel::bplapply()]

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
head(res)
```

---

get_domain_gene_for_enrichment

Extract Domain-Altering Genes for Enrichment

Description

Return genes whose inclusion/exclusion isoforms show unique protein domain gain or loss.

Usage

```
get_domain_gene_for_enrichment(hits)
```

Arguments

hits	Data frame with domain-annotation columns.

Value

Character vector of gene IDs.

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
annotation_df <- load_example_data("annotation_df")$annotation_df
matched <- get_matched_events_chunked(res, annotation_df$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annotation_df$sequences)
pairs <- get_pairs(x_seq, source="multi")
seq_compare <-compare_sequence_frame(pairs, annotation_df$annotations)
annotation_df <- get_annotation(load = 'test')
interpro_features <- get_protein_features(c("interpro"), annotation_df$annotations, timeout = 600, test = TRUE)
protein_feature_total <- get_comprehensive_annotations(list(interpro_features))

exon_features <- get_exon_features(annotation_df$annotations, protein_feature_total)
```

```
hits_domain <- get_domains(seq_compare, exon_features)

bg <- get_background(source = "hit_index",
                     input = sample_frame,
                     annotations = annotation_df$annotations,
                     protein_features = protein_feature_total)
enrichment <- get_enrichment(get_domain_gene_for_enrichment(hits_domain), bg$gene_id, species = 'human', 'ensembl
print(enrichment)
```

`get_domains` *Add protein domain annotations to splicing events*

Description

Annotates each splicing event with protein domains that are gained, lost, or uniquely present in inclusion or exclusion isoforms.

Usage

```
get_domains(
  hits,
  exon_features,
  show_protein_domains = FALSE,
  return_class = c("auto", "data.table", "S4")
)
```

Arguments

`hits`	'data.frame', 'data.table', or 'SpliceImpactResult' containing transcript pairs with at least 'transcript_id_case', 'transcript_id_control', 'exons_case', 'exons_control', and 'event_type'.
`exon_features`	'data.frame' of exon-domain annotations with columns 'ensembl_transcript_id', 'ensembl_peptide_id', 'exon_id', 'database', 'feature_id', 'name', 'overlap_aa_start', 'overlap_aa_end'.
`show_protein_domains`	Logical; if 'TRUE', include full protein-level domain sets ('domains_protein_case' / 'domains_protein_control').
`return_class`	Character. Output mode: '"data.table"', '"S4"', or '"auto"' (default). In 'auto', S4 input returns updated S4 output.

Details

Internally, this function builds a domain lookup table from an exon feature annotation (e.g. InterPro, Pfam) and extracts per-exon and per-transcript domain lists for each isoform in 'hits'. Differences between the inclusion ('*_case') and exclusion ('*_control') isoforms are then summarized as:

* ‘case_only_domains‘: domains unique to the inclusion isoform * ‘control_only_domains‘: domains unique to the exclusion isoform * ‘diff_n‘: total number of non-shared domains

If ‘show_protein_domains = TRUE‘, additional columns report full domain sets across the entire inclusion/exclusion proteins.

Value

The input ‘hits‘ table with added columns (or updated ‘SpliceImpactResult‘ when ‘return_class‘ resolves to S4):

- ‘domains_exons_case‘, ‘domains_exons_control‘ domains found on event exons
- ‘case_only_domains‘, ‘control_only_domains‘ domains unique to each isoform
- ‘case_only_domains_list‘, ‘control_only_domains_list‘, ‘either_domains_list‘ list-columns
- ‘case_only_n‘, ‘control_only_n‘, ‘diff_n‘ domain counts
- optionally, ‘domains_protein_case‘ / ‘domains_protein_control‘

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
annotation_df <- load_example_data("annotation_df")$annotation_df
matched <- get_matched_events_chunked(res, annotation_df$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annotation_df$sequences)
pairs <- get_pairs(x_seq, source="multi")
seq_compare <-compare_sequence_frame(pairs, annotation_df$annotations)
interpro_features <- get_protein_features(c("interpro"), annotation_df$annotations, timeout = 600, test = TRUE)
protein_feature_total <- get_comprehensive_annotations(list(interpro_features))

exon_features <- get_exon_features(annotation_df$annotations, protein_feature_total)

hits_domain <- get_domains(seq_compare, exon_features)
print(hits_domain)
```

---

get_enrichment *Gene Set Enrichment for Splicing-Linked Gene Lists*

Description

Perform over-representation analysis for a foreground gene set, optionally against a background universe, using GO, MSigDB, and Reactome categories.

Usage

```
get_enrichment(
  foreground,
  background = NULL,
  species = c("human", "mouse"),
  gene_id_type = c("symbol", "ensembl"),
  sources = c("GO:BP", "GO:MF", "GO:CC", "MSigDB:H", "MSigDB:C2:CP:REACTOME"),
  min_size = 10,
  max_size = 2000,
  p_adjust_cutoff = 0.05,
  simplify_go = TRUE,
  top_n_plot = 20,
  plot_type = c("dot", "bar")
)
```

Arguments

foreground	Character vector of gene IDs (symbols or Ensembl IDs).
background	Optional character vector of background genes (universe). If 'NULL', all genes present in annotation collections are used.
species	Species for enrichment catalog ('"human"' or '"mouse"').
gene_id_type	Type of input gene IDs ('"symbol"' or '"ensembl"').
sources	Character vector selecting enrichment sources. Example options include: • '"GO:BP"', '"GO:MF"', '"GO:CC"' • '"MSigDB:H"', '"MSigDB:C2:CP:REACTOME"'
min_size	Minimum term size (default '10').
max_size	Maximum term size (default '2000').
p_adjust_cutoff	FDR cutoff for reporting significant terms.
simplify_go	Whether to apply GO term redundancy reduction.
top_n_plot	Number of terms to visualize in the quick plot.
plot_type	'"dot"' (default) or '"bar"'.

Details

This is a convenience wrapper around 'clusterProfiler', 'msigdbr', and optionally 'ReactomePA', producing a combined enrichment table and a quick visualization of top significant terms.

Input genes are internally mapped to Entrez IDs. Enrichment tests are performed using:

* 'clusterProfiler::enrichGO' * 'clusterProfiler::enricher' (MSigDB) * 'ReactomePA::enrichPathway' (optional)

Value

A list with:

results_per_source List of enrichment result tables per source

results_combined Combined enrichment table

results_signif Filtered table by FDR cutoff

plot A 'ggplot2' object visualizing top terms

Note

Requires these packages installed: 'clusterProfiler', 'msigdbr', 'data.table', 'AnnotationDbi', 'ggplot2', and 'org.Hs.eg.db' or 'org.Mm.eg.db'. For Reactome analysis you must also install 'ReactomePA'.

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)

annotation_df <- load_example_data("annotation_df")$annotation_df
interpro_features <- get_protein_features(c("interpro"), annotation_df$annotations, timeout = 600, test = TRUE)
protein_feature_total <- get_comprehensive_annotations(list(interpro_features))

exon_features <- get_exon_features(annotation_df$annotations, protein_feature_total)

matched <- get_matched_events_chunked(res, annotation_df$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annotation_df$sequences)
pairs <- get_pairs(x_seq, source="multi")
seq_compare <-compare_sequence_frame(pairs, annotation_df$annotations)
hits_domain <- get_domains(seq_compare, exon_features)

bg <- get_background(source = "hit_index",
                     input = sample_frame,
                     annotations = annotation_df$annotations,
                     protein_features = protein_feature_total)
enrichment <- get_enrichment(res$gene_id, bg$gene_id, species = 'human', 'ensembl', 'MSigDB:H')
print(enrichment)
```

---

get_exon_features *Map protein features to coding exons*

Description

Overlaps amino acid-based protein features (e.g., InterPro, TMHMM) with exon coding spans defined by transcript-relative CDS coordinates.

Usage

```
get_exon_features(gtf_dt, feat, inclusive = TRUE)
```

Arguments

gtf_dt	A data.frame or data.table containing transcript and exon annotation, including type, transcript_id, cds_rel_start, and cds_rel_stop columns (see [add_exon_coding_information()]).
feat	A data.frame or data.table of long-format protein features (from [get_protein_features()] or [get_comprehensive_annotations()]) containing columns ensembl_transcript_id, start, stop, database, feature_id, name, and alt_name.
inclusive	Logical; whether to round both feature and exon boundaries upward when converting from nucleotide to amino acid coordinates (default TRUE). If FALSE, downstream exons own partial codons to avoid double counting.

Value

A data.table of overlapping feature-exon pairs with amino acid coordinates (overlap_aa_start, overlap_aa_end, overlap_aa_len) and associated exon and feature metadata.

Examples

```
annotation_df <- get_annotation(load = 'test')
interpro_features <- get_protein_features(c("interpro"), annotation_df$annotations, timeout = 600, test = TRUE)
protein_feature_total <- get_comprehensive_annotations(list(interpro_features))

exon_features <- get_exon_features(annotation_df$annotations, protein_feature_total)
print(exon_features)
```

---

get_gene_enrichment *Unified gene selector for enrichment foregrounds*

Description

Selects a foreground gene vector for enrichment from DI results (‘res‘) or hits-final-like tables (‘hits‘) using a single wrapper.

Usage

```
get_gene_enrichment(
  mode = c("di", "ppi", "domain"),
  x = NULL,
  res = NULL,
  hits = NULL,
  padj_threshold = 0.05,
  delta_psi_threshold = 0.1
)
```

Arguments

mode	One of ‘"di"‘, ‘"ppi"‘, ‘"domain"‘.
x	Optional generic input. For ‘mode="di"‘, this should be ‘res‘-like; for ‘mode="ppi"‘/‘"domain"‘, this should be hits-final-like. If ‘x‘ is S4, the appropriate slot is used automatically.
res	Optional DI table (or S4) used when ‘mode="di"‘. If provided, it is preferred over ‘x‘.
hits	Optional hits-final-like table (or S4) used when ‘mode="ppi"‘/‘"domain"‘. If provided, it is preferred over ‘x‘.
padj_threshold	FDR cutoff used only for ‘mode="di"‘.
delta_psi_threshold	Absolute delta-psi cutoff used only for ‘mode="di"‘.

Details

- ‘mode = "di"‘ uses differential inclusion columns (‘gene_id‘, ‘padj‘, ‘delta_psi‘) - ‘mode = "ppi"‘ uses hits-final-like columns (‘gene_id‘, ‘n_ppi‘) - ‘mode = "domain"‘ uses hits-final-like columns (‘gene_id‘, ‘diff_n‘)

Inputs can be a ‘data.table‘/‘data.frame‘ or a ‘SpliceImpactResult‘ S4 object.

Value

Character vector of unique gene IDs.

Examples

```
res <- data.table::data.table(
  gene_id = c("ENSG000001", "ENSG000001", "ENSG000002"),
  padj = c(0.01, 0.20, 0.03),
  delta_psi = c(0.25, 0.05, -0.30)
)
hits <- data.table::data.table(
  gene_id = c("ENSG000001", "ENSG000002", "ENSG000003"),
  n_ppi = c(1L, 0L, 2L),
  diff_n = c(0L, 1L, 2L)
)
get_gene_enrichment(mode = "di", res = res)
get_gene_enrichment(mode = "ppi", hits = hits)
get_gene_enrichment(mode = "domain", hits = hits)

hits_s4 <- data.table::data.table(
  event_id = c("E1", "E2", "E3"),
  event_type = c("SE", "A3SS", "MXE"),
  gene_id = c("ENSG000001", "ENSG000002", "ENSG000003"),
  chr = c("chr1", "chr1", "chr2"),
  strand = c("+", "+", "-"),
  transcript_id_control = c("TX1", "TX3", "TX5"),
  transcript_id_case = c("TX2", "TX4", "TX6"),
  inc_control = c("100-110", "200-210", "300-310"),
```

```
  inc_case = c("100-115", "205-215", "305-315"),
  exc_control = c("120-130", "220-230", "320-330"),
  exc_case = c("121-130", "225-235", "325-335"),
  n_ppi = c(1L, 0L, 2L),
  diff_n = c(0L, 1L, 2L)
)
obj <- as_splice_impact_result(hits_final = hits_s4)
get_gene_enrichment(mode = "ppi", x = obj)
get_gene_enrichment(mode = "domain", x = obj)
```

`get_hitindex` *Load HIT index PSI files for one or more samples/conditions.*

Description

Load HIT index PSI files for one or more samples/conditions.

Usage

```
get_hitindex(paths_df, keep_annotated_first_last = FALSE)
```

Arguments

`paths_df`	Data.frame with columns: `path`, `condition`, and optionally `sample_name`.
`keep_annotated_first_last`	Logical; if TRUE, retain only annotated first/last exons and normalize PSI.

Value

A standardized ‘data.table‘ of HIT index PSI values with inclusion/exclusion and metadata.

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame, keep_annotated_first_last = TRUE)
print(hit_index)
```

get_hits_core *Convenience accessor for core paired-hit columns*

Description

Convenience accessor for core paired-hit columns

Usage

```
get_hits_core(x, drop_missing = TRUE, keep_internal_keys = FALSE)
```

Arguments

x	A [SpliceImpactResult] object or a paired-hits ‘data.frame‘/‘data.table‘.
drop_missing	Logical; if ‘TRUE‘, silently drops requested columns that are absent. If ‘FALSE‘, errors on missing columns.
keep_internal_keys	Passed through when ‘x‘ is S4. Default ‘FALSE‘.

Value

‘data.table‘ with the ‘core‘ subset. Works for both S4 and paired-hits ‘data.table‘ input.

Examples

```
hits <- data.table::data.table(
  event_id = c("E1", "E2"),
  event_type = c("SE", "A3SS"),
  gene_id = c("ENSG000001", "ENSG000002"),
  chr = c("chr1", "chr2"),
  strand = c("+", "-"),
  transcript_id_control = c("TX1", "TX3"),
  transcript_id_case = c("TX2", "TX4"),
  n_ppi = c(1L, 0L),
  diff_n = c(1L, 0L)
)
print(get_hits_core(hits))
```

get_hits_domain *Convenience accessor for domain-focused paired-hit columns*

Description

Convenience accessor for domain-focused paired-hit columns

Usage

```
get_hits_domain(x, drop_missing = TRUE, keep_internal_keys = FALSE)
```

Arguments

x	A [SpliceImpactResult] object or a paired-hits ‘data.frame‘/‘data.table‘.
drop_missing	Logical; if ‘TRUE‘, silently drops requested columns that are absent. If ‘FALSE‘, errors on missing columns.
keep_internal_keys	Passed through when ‘x‘ is S4. Default ‘FALSE‘.

Value

‘data.table‘ with the ‘domain‘ subset. Works for both S4 and paired-hits ‘data.table‘ input.

Examples

```
hits <- data.table::data.table(
  event_id = c("E1", "E2"),
  event_type = c("SE", "A3SS"),
  gene_id = c("ENSG000001", "ENSG000002"),
  chr = c("chr1", "chr2"),
  strand = c("+", "-"),
  transcript_id_control = c("TX1", "TX3"),
  transcript_id_case = c("TX2", "TX4"),
  case_only_domains = c("IPR0001", ""),
  control_only_domains = c("", "IPR0002"),
  case_only_n = c(1L, 0L),
  control_only_n = c(0L, 1L),
  diff_n = c(1L, 1L)
)
print(get_hits_domain(hits))
```

---

get_hits_final_view *Access paired-hits as compact data.table subsets*

Description

Extracts ‘paired_hits‘ columns from a [SpliceImpactResult] (or a paired-hits ‘data.table‘) using predefined subset groups such as ‘core‘, ‘domain‘, ‘ppi‘, and ‘sequence‘.

Usage

```
get_hits_final_view(
  x,
  col_subset = c("core"),
  cols = NULL,
  drop_missing = TRUE,
  keep_internal_keys = FALSE
)
```

Arguments

x	A [SpliceImpactResult] object or a paired-hits ‘data.frame‘/‘data.table‘.
col_subset	Character vector of subset names. Any of ‘"core"‘, ‘"domain"‘, ‘"ppi"‘, ‘"sequence"‘, or ‘"all"‘.
cols	Optional explicit column vector. If supplied, ‘col_subset‘ is ignored.
drop_missing	Logical; if ‘TRUE‘, silently drops requested columns that are absent. If ‘FALSE‘, errors on missing columns.
keep_internal_keys	Passed through when ‘x‘ is S4. Default ‘FALSE‘.

Value

A ‘data.table‘ containing the selected columns. Row count and row order are preserved from the input (‘SpliceImpactResult@paired_hits‘ or provided ‘data.table‘).

Examples

```
hits <- data.table::data.table(
  event_id = c("E1", "E2"),
  event_type = c("SE", "A3SS"),
  gene_id = c("ENSG000001", "ENSG000002"),
  chr = c("chr1", "chr2"),
  strand = c("+", "-"),
  transcript_id_control = c("TX1", "TX3"),
  transcript_id_case = c("TX2", "TX4"),
  protein_id_control = c("P1", "P3"),
  protein_id_case = c("P2", "P4"),
  inc_control = c("100-110", "200-210"),
  inc_case = c("100-115", "205-215"),
  exc_control = c("120-130", "220-230"),
  exc_case = c("121-130", "225-235"),
  case_only_domains = c("IPR0001", ""),
  control_only_domains = c("", "IPR0002"),
  case_only_n = c(1L, 0L),
  control_only_n = c(0L, 1L),
  diff_n = c(1L, 1L),
  case_ppi = c("A;B", "C"),
  control_ppi = c("A", "C;D"),
  n_case_ppi = c(2L, 1L),
  n_control_ppi = c(1L, 2L),
  n_ppi = c(1L, 1L),
  dna_pid = c(0.95, 0.90),
  prot_pid = c(0.90, 0.85),
  frame_call = c("Match", "Frameshift")
)
print(get_hits_final_view(hits, col_subset = c("core", "ppi")))
```

get_hits_ppi *Convenience accessor for PPI-focused paired-hit columns*

Description

Convenience accessor for PPI-focused paired-hit columns

Usage

```
get_hits_ppi(x, drop_missing = TRUE, keep_internal_keys = FALSE)
```

Arguments

x	A [SpliceImpactResult] object or a paired-hits ‘data.frame‘/‘data.table‘.
drop_missing	Logical; if ‘TRUE‘, silently drops requested columns that are absent. If ‘FALSE‘, errors on missing columns.
keep_internal_keys	Passed through when ‘x‘ is S4. Default ‘FALSE‘.

Value

‘data.table‘ with the ‘ppi‘ subset. Works for both S4 and paired-hits ‘data.table‘ input.

Examples

```
hits <- data.table::data.table(
  event_id = c("E1", "E2"),
  event_type = c("SE", "A3SS"),
  gene_id = c("ENSG000001", "ENSG000002"),
  chr = c("chr1", "chr2"),
  strand = c("+", "-"),
  transcript_id_control = c("TX1", "TX3"),
  transcript_id_case = c("TX2", "TX4"),
  case_ppi = c("A;B", "C"),
  control_ppi = c("A", "C;D"),
  n_case_ppi = c(2L, 1L),
  n_control_ppi = c(1L, 2L),
  n_ppi = c(1L, 1L)
)
print(get_hits_ppi(hits))
```

`get_hits_sequence` *Convenience accessor for sequence/frame-focused paired-hit columns*

Description

Convenience accessor for sequence/frame-focused paired-hit columns

Usage

```
get_hits_sequence(x, drop_missing = TRUE, keep_internal_keys = FALSE)
```

Arguments

`x`	A [SpliceImpactResult] object or a paired-hits ‘data.frame‘/‘data.table‘.
`drop_missing`	Logical; if ‘TRUE‘, silently drops requested columns that are absent. If ‘FALSE‘, errors on missing columns.
`keep_internal_keys`	Passed through when ‘x‘ is S4. Default ‘FALSE‘.

Value

‘data.table‘ with the ‘sequence‘ subset. Works for both S4 and paired-hits ‘data.table‘ input.

Examples

```
hits <- data.table::data.table(
  event_id = c("E1", "E2"),
  event_type = c("SE", "A3SS"),
  gene_id = c("ENSG000001", "ENSG000002"),
  chr = c("chr1", "chr2"),
  strand = c("+", "-"),
  transcript_id_control = c("TX1", "TX3"),
  transcript_id_case = c("TX2", "TX4"),
  protein_id_control = c("P1", "P3"),
  protein_id_case = c("P2", "P4"),
  dna_pid = c(0.95, 0.90),
  prot_pid = c(0.90, 0.85),
  frame_call = c("Match", "Frameshift")
)
print(get_hits_sequence(hits))
```

`get_manual_features` *Incorporate user-supplied protein features*

Description

Converts a manual feature table into the standardized long format and optionally merges it with biomaRt-derived features.

Usage

```
get_manual_features(
  manual_features,
  gtf_df,
  biomaRt_features = NULL,
  load_path = NULL,
  save_path = NULL
)
```

Arguments

`manual_features`	Data.frame or data.table with at least `name`, `start`, `stop` amino acid, and one of `ensembl_transcript_id` or `ensembl_peptide_id`.
`gtf_df`	get_annotation annotation output
`biomaRt_features`	Optional `data.table` of features from [get_protein_features()] to merge with.
`load_path`	Optional path to load precomputed manual features.
`save_path`	Optional path to save the combined feature table.

Value

A `data.table` of manual (and optionally combined) protein features.

See Also

[add_user_features()]

Examples

```
annotation_df <- load_example_data("annotation_df")$annotation_df
user_df <- data.frame(
  ensembl_transcript_id = c(
    "ENST00000511072","ENST00000374900","ENST00000373020","ENST00000456328",
    "ENST00000367770","ENST00000331789","ENST00000335137","ENST00000361567",
    NA,                    "ENST00000380152"
  ),
  ensembl_peptide_id = c(
```

```
    "ENSP00000426975", NA,                    "ENSP00000362048","ENSP00000407743",
    "ENSP00000356802","ENSP00000326734", NA,                   "ENSP00000354587",
    "ENSP00000364035", NA
  ),
  name = c(
    "Low complexity","Transmembrane helix","Coiled-coil","Signal peptide",
    "Transmembrane helix","Low complexity","Coiled-coil","Transmembrane helix",
    "Signal peptide","Low complexity"
  ),
  start = c(80L, 201L, 35L, 1L, 410L, 150L, 220L, 30L, 1L, 300L),
  stop  = c(120L,223L, 80L, 20L, 430L, 190L, 260L, 55L, 24L, 360L),
 database  = c("seg","tmhmm","ncoils","signalp","tmhmm","seg","ncoils","tmhmm","signalp", NA),
 alt_name  = c(NA,"TMhelix",NA,"SignalP-noTM", "TMhelix", NA, NA, "TMhelix", "SignalP-TAT", NA),
  feature_id = c(NA, NA, NA, NA, NA, NA, NA, NA, NA, NA)
)
user_features <- get_manual_features(user_df, annotation_df$annotations)
print(user_features)
```

`get_matched_events_chunked` *Match splicing events to transcript annotations in chunks*

Description

This is a wrapper around match_events_to_annotations_vec that processes large event tables in manageable chunks to reduce memory usage. It sequentially runs matching per chunk and concatenates the results.

Usage

```
get_matched_events_chunked(
  events,
  annotations,
  chunk_size = 50000,
  minOverlap = 0.05,
  return_class = c("auto", "data.table", "S4")
)
```

Arguments

`events`	A data.frame or data.table of event definitions. Must include coordinates compatible with match_events_to_annotations_vec
`annotations`	A data.frame of transcript/exon annotation rows. Typically generated by get_annotations
`chunk_size`	Integer, number of event rows to process per chunk (default = 50,000).
`minOverlap`	double ranging from 0 to 1 (default 0.05), required minimum overlap to consider a match
`return_class`	Character. Output mode: ‘"data.table"‘, ‘"S4"‘, or ‘"auto"‘ (default). In ‘auto‘, S4 input returns updated S4 output.

Details

This function is intended for large-scale event matching across many splicing events, where running the full table at once may exceed memory limits. It can later be parallelized using **future.apply** or similar.

Value

A data.table with matched transcripts and exons for all events. The output order matches the original event order.

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
print(sample_frame)

hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
annots <- load_example_data("annotation_df")$annotation_df
matched <- get_matched_events_chunked(res, annots$annotations, chunk_size = 2000)
print(matched)
```

---

get_pairs *Pair inclusion and exclusion forms of splicing events*

Description

Builds paired tables of inclusion/exclusion forms for splicing events from rMATS-like or HITindex-like inputs. In rMATS mode, events are paired when both INC and EXC forms exist for a given event ID. In HITindex mode, all positive and negative deltaPSI rows within each event are cross-joined.

Usage

```
get_pairs(
  x,
  source = c("paired", "multi"),
  return_class = c("auto", "data.table", "S4")
)
```

Arguments

x	A data.frame, data.table, or 'SpliceImpactResult' containing splicing event information.
source	Character string specifying input structure:
	"paired" (rMATS-like) requires exactly one INC and one EXC per event ID.

	`"multi"` (HITindex-like) pairs all positive and negative `delta_psi` values within each event.
`return_class`	Character. Output mode: ‘"data.table"‘, ‘"S4"‘, or ‘"auto"‘ (default). In ‘auto‘, S4 input returns updated S4 output.

Details

In `source="paired"` mode, only events with exactly one INC and one EXC row are retained. In `source="multi"` mode, all positive deltaPSI rows are joined with all negative deltaPSI rows (cartesian join) within each event.

Value

A data.table (or updated ‘SpliceImpactResult‘ when ‘return_class‘ resolves to S4) where each row represents an inclusion-exclusion pair of the same event.

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
annots <- load_example_data("annotation_df")$annotation_df
matched <- get_matched_events_chunked(res, annots$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annots$sequences)
pairs <- get_pairs(x_seq, source="multi")
print(pairs)
```

---

get_ppi_gene_enrichment

Extract Protein-Interaction-Affected Genes for Enrichment

Description

Return genes whose splicing affects known protein-protein interactions.

Usage

```
get_ppi_gene_enrichment(hits)
```

Arguments

`hits`	Data frame with PPI annotation columns.

Value

Character vector of gene IDs.

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
annotation_df <- load_example_data("annotation_df")$annotation_df
matched <- get_matched_events_chunked(res, annotation_df$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annotation_df$sequences)
pairs <- get_pairs(x_seq, source="multi")
seq_compare <-compare_sequence_frame(pairs, annotation_df$annotations)
annotation_df <- get_annotation(load = 'test')
interpro_features <- get_protein_features(c("interpro"), annotation_df$annotations, timeout = 600, test = TRUE)
protein_feature_total <- get_comprehensive_annotations(list(interpro_features))

exon_features <- get_exon_features(annotation_df$annotations, protein_feature_total)

hits_domain <- get_domains(seq_compare, exon_features)
ppi <- get_ppi_interactions()
hits_final <- get_ppi_switches(hits_domain, ppi, protein_feature_total)
bg <- get_background(source = "hit_index",
                     input = sample_frame,
                     annotations = annotation_df$annotations,
                     protein_features = protein_feature_total)
enrichment <- get_enrichment(get_ppi_gene_enrichment(hits_final), bg$gene_id, species = 'human', 'ensembl', 'MSig
print(enrichment)
```

---

get_ppi_interactions *Pull PPI from SpliceImpactR's data*

Description

Generation details in inst/scripts

Usage

```
get_ppi_interactions()
```

Value

A 'data.table' of interaction edges used by PPI switching utilities.

Examples

```
ppi_int <- get_ppi_interactions()
print(ppi_int)
```

get_ppi_switches *Annotate hits_domain with PPI changes for inclusion vs exclusion forms*

Description

Adds list-cols case_ppi/control_ppi (partner genes) plus counts. Also returns (optionally useful) per-event token sets in PFAM + ELM forms.

Usage

```
get_ppi_switches(
  hits_domain,
  ppi,
  protein_feature_total,
  return_class = c("auto", "data.table", "S4")
)
```

Arguments

hits_domain	data.table with gene_id and list-cols case_only_domains_list / control_only_domains_list
ppi	wide interaction table from saved data (get_ppi)
protein_feature_total	table with database/clean_name/feature_id for interpro mapping
return_class	Character. Output mode: ‘"data.table"‘, ‘"S4"‘, or ‘"auto"‘ (default). In ‘auto‘, S4 input returns updated S4 output.

Value

A ‘data.table‘ identical to ‘hits_domain‘ with added columns (or updated ‘SpliceImpactResult‘ when ‘return_class‘ resolves to S4):

‘case_ppi‘, ‘control_ppi‘ Lists of partner transcripts unique to inclusion or exclusion isoforms.

‘n_control_ppi‘, ‘n_control_ppi‘ Counts of gained/lost interactions.

‘n_ppi‘ Total PPI changes (sum of both directions).

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
annotation_df <- load_example_data("annotation_df")$annotation_df
matched <- get_matched_events_chunked(res, annotation_df$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annotation_df$sequences)
pairs <- get_pairs(x_seq, source="multi")
seq_compare <-compare_sequence_frame(pairs, annotation_df$annotations)
```

```
annotation_df <- get_annotation(load = 'test')
interpro_features <- get_protein_features(c("interpro"), annotation_df$annotations, timeout = 600, test = TRUE)
protein_feature_total <- get_comprehensive_annotations(list(interpro_features))

exon_features <- get_exon_features(annotation_df$annotations, protein_feature_total)

hits_domain <- get_domains(seq_compare, exon_features)

bg <- get_background(source = "hit_index",
                     input = sample_frame,
                     annotations = annotation_df$annotations,
                     protein_features = protein_feature_total)
ppi <- get_ppi_interactions()
hits_ppi <- get_ppi_switches(hits_domain, ppi, protein_feature_total)
print(hits_ppi)
hits_ppi[n_ppi > 0, .(event_id, gene_id, n_case_ppi, n_control_ppi, n_ppi, case_ppi, control_ppi)]
```

`get_protein_features` *External function to fetch protein features from biomaRt*

Description

Here we also remove any duplicate and overlapping domains We also add the genomic location to the name of the protein feature for downstream safeguarding and precision. This is to prevent different occurences of the same domain being called as the same in domain identification and enrichment.

Usage

```
get_protein_features(
 biomaRt_databases = c("interpro", "mobidblite", "seg", "ncoils", "tmhmm", "signalp",
    "elm", "gene3d", "pfam"),
  gtf_df,
  sequences = NULL,
  load_path = NULL,
  save_path = NULL,
  base_dir = NULL,
  use_cache = TRUE,
  force_refresh = FALSE,
  timeout = 600,
  ensembl_mirror = NULL,
  species = c("human", "mouse"),
  release = 109,
  test = FALSE,
  combine_overlaps = FALSE
)
```

Arguments

biomaRt_databases	choose what biomaRt attribute to access, defaulting to interpro, mobidblite, seg, ncoils, tmhmm, signalp
gtf_df	annotations from get_annotation()
sequences	only necessary if loading SLiMs from elm get_annotation() (default "sequences") output
load_path	path to load prior protein features from
save_path	path to save prior protein features from
base_dir	Optional cache root. If 'NULL' (default), uses package cache under 'tools::R_user_dir("SpliceImpactR", "cache")'.
use_cache	Logical; if 'TRUE' (default), cache and reuse final 'get_protein_features()' outputs through BiocFileCache.
force_refresh	Logical; if 'TRUE', recompute and overwrite any existing BiocFileCache entry for this parameter/input signature.
timeout	ability to extend timeout if biomaRt is not cooperating
ensembl_mirror	Optional Ensembl mirror to try first for BioMart connections; one of '"useast"', '"www"', or '"asia"'. If 'NULL', mirrors are tried in fallback order when 'release = NULL'. If a specific 'release' is provided, biomaRt ignores mirror selection.
species	Character string giving the Ensembl BioMart dataset (default "human"). For mouse, use "mouse".
release	Release version from Ensembl associated with the GENCODE version used in [get_annotation()]. See the GENCODE human/mouse release listings to map GENCODE and Ensembl versions.
test	Logical; bool for whether to load from reduced test set.
combine_overlaps	simplifies protein feature output and combines protein features with the same ID and overlapping coords. Sometimes not desireable

Value

A 'data.table' with one row per protein feature and transcript coupling

Examples

```
annotation_df <- load_example_data("annotation_df")$annotation_df
interpro_features <- get_protein_features(c("interpro"), annotation_df$annotations, annotation_df$sequences, tim
print(interpro_features)
```

`get_proximal_shift_from_hits` *Classify splicing events as proximal or distal*

Description

Determines whether inclusion/exclusion events correspond to proximal or distal terminal exon usage for **AFE** (Alternative First Exon) and **ALE** (Alternative Last Exon) events, based on genomic coordinates and strand orientation.

Usage

```
get_proximal_shift_from_hits(hits)
```

Arguments

`hits`	‘data.frame‘ or ‘data.table‘ containing at least: • ‘event_id‘ • ‘event_type‘ • ‘strand‘ • ‘inc_case‘, ‘inc_control‘ • ‘delta_psi_case‘, ‘delta_psi_control‘

Details

The function compares the genomic coordinates of the inclusion (‘inc_case‘) and exclusion (‘inc_control‘) segments per event:

- For **AFE** events, proximal = exon with smaller start coordinate on the ‘+‘ strand (or larger end on ‘-‘ strand).
- For **ALE** events, proximal = exon with smaller start coordinate on the ‘+‘ strand (or larger end on ‘-‘ strand).

Events outside these types are labeled ‘"nonTerminal"‘.

If `plot = TRUE`, a summary donut chart is printed showing the proportion of proximal vs distal usage per event type.

Value

A ‘data.table‘ identical to ‘hits‘ with an additional column (named ‘V1‘) specifying ‘"proximal"‘, ‘"distal"‘, ‘"overlap"‘, or ‘"nonTerminal"‘.

See Also

[plot_prox_dist()]

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
annotation_df <- load_example_data("annotation_df")$annotation_df
matched <- get_matched_events_chunked(res, annotation_df$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annotation_df$sequences)
pairs <- get_pairs(x_seq, source="multi")
proximal_output <- get_proximal_shift_from_hits(pairs)
print(proximal_output)
```

`get_rmats` *Expand rMATS event tables into scalar exon inclusion/exclusion coordinates.*

Description

Converts rMATS "event" tables (SE, MXE, A3SS, A5SS, RI) into standardized scalar representations with explicit inclusion/exclusion segments for downstream genomic mapping.

Usage

```
get_rmats(DT)
```

Arguments

`DT`	A ‘data.table‘ or ‘data.frame‘ of rMATS output (merged or per-sample).

Details

Handles all five canonical rMATS event types (SE, MXE, A3SS, A5SS, RI), applying strand-aware logic for MXE and coordinate adjustments for A3/A5. Non-standard columns (e.g. IJC_SAMPLE_1) are checked for presence.

Value

A standardized ‘data.table‘ containing:

- event_id unique event identifier.
- event_type rMATS event type (SE, MXE, etc.).
- form inclusion/exclusion form.
- gene_id, chr, strand.
- inc, exc scalar genomic segments (string: e.g. ‘"100-200;300-400"‘).
- inclusion_reads, exclusion_reads, psi - numeric metrics.
- condition, sample, source_file - carried forward if present.

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
rmats <- get_rmats(load_rmats(sample_frame, use = "JCEC", event_types = c("MXE", "SE", "A3SS", "A5SS", "RI")))
print(rmats)
```

`get_rmats_hit` *Wrapper function to get both rmats and hit index cleanly*

Description

Wrapper function to get both rmats and hit index cleanly

Usage

```
get_rmats_hit(
  sample_frame,
  event_types = c("ALE", "AFE", "MXE", "SE", "A3SS", "A5SS", "RI"),
  use = "JCEC",
  keep_annotated_first_last = TRUE
)
```

Arguments

`sample_frame`	Data.frame with columns: `path`, `condition`, and `sample_name`.
`event_types`	event types to load from rMATS
`use`	Character scalar, one of `"JC"` or `"JCEC"`.
`keep_annotated_first_last`	Logical; if TRUE, retain only annotated first/last exons and normalize PSI.

Value

a 'data.table' for all the event types desired from the paths supplied contains: event_id (unique id for event), event_type (AS event type), form (INC/EXC), gene_id (ensembl id), strand, inc, exc (inclusion and exclusion coords) inclusion reads, exclusion reads, psi, sample, condition, source file

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
data <- get_rmats_hit(sample_frame, event_types = c("ALE", "AFE", "MXE", "SE", "A3SS", "A5SS", "RI"))
print(data)
```

`get_rmats_post_di` *Import post-differential-inclusion rMATS results*

Description

This function reads post-DI rMATS results and converts them into the standardized SpliceImpactR long format with one row per event x (INC/EXC) form.

Input can be: * a data.frame with columns ‘path‘, ‘grp1‘, ‘grp2‘, ‘event_type‘, in which case each file is read and processed; or * a single rMATS results data.frame, in which case ‘event_type‘ must be supplied.

For each event, the function constructs paired INC and EXC entries: * ‘inc‘ contains genomic segments included in the form * ‘exc‘ contains the excluded segment(s) * ‘delta_psi‘, ‘p.value‘, and ‘padj‘ are assigned using the rMATS-reported values

Event IDs are automatically generated (event_type:N) if not supplied

Usage

```
get_rmats_post_di(input, event_type = NULL)
```

Arguments

`input`	Either: * a data.frame with columns ‘path‘, ‘grp1‘, ‘grp2‘, ‘event_type‘, or * a data.frame of rMATS post-DI results.
`event_type`	Optional event type when ‘input‘ contains a single rMATS data.frame. Ignored when file metadata table is supplied.

Value

A ‘data.table‘ with columns:

event_id unique event identifier

event_type splicing event type

form `"INC"` or `"EXC"`

gene_id gene ID

chr chromosome

strand strand

inc genomic coordinates of included segment(s)

exc genomic coordinates of excluded segment(s)

p.value rMATS p-value

padj FDR

delta_psi signed PSI change (+INC, -EXC)

Examples

```
# # Multiple files
# input <- data.frame(
#   path = c('/path/A3SS.MATS.JC.txt', '/path2/A5SS.MATS.JC.txt'),
#   grp1 = c("WT","WT"),
#   grp2 = c("KO","KO"),
#   event_type = c("A3SS", "A5SS")
# )
# res <- get_rmats_post_di(meta)

# Single rMATS table already loaded as df
df <- data.frame(
  ID = 1L,
  GeneID = "ENSG00000182871",
  geneSymbol = "COL18A1",
  chr = "chr21",
  strand = "+",
  longExonStart_0base = 45505834L,
  longExonEnd = 45505966L,
  shortES = 45505837L,
  shortEE = 45505966L,
  flankingES = 45505357L,
  flankingEE = 45505431L,
  ID.2 = 2L,
  IJC_SAMPLE_1 = "4,1,0",
  SJC_SAMPLE_1 = "9,12,3",
  IJC_SAMPLE_2 = "0,4,5",
  SJC_SAMPLE_2 = "11,15,15",
  IncFormLen = 52L,
  SkipFormLen = 49L,
  PValue = 0.6967562,
  FDR = 1,
  IncLevel1 = "0.295,0.073,0.0",
  IncLevel2 = "0.0,0.201,0.239",
  IncLevelDifference = -0.024,
  stringsAsFactors = FALSE
)
res2 <- get_rmats_post_di(df, event_type = "A3SS")
print(res2)
```

---

get_splicing_impact *End-to-end SpliceImpactR wrapper with selectable output class*

Description

Runs the core SpliceImpactR pipeline from raw event table (or sample paths) through paired domain/PPI calls, then returns either a compact ‘data.table‘ bundle (‘data‘, ‘res‘, ‘hits_final‘) or an S4 [SpliceImpactResult].

Usage

```
get_splicing_impact(
  sample_frame = NULL,
  data = NULL,
  res = NULL,
  annotation_df = NULL,
  protein_feature_total = NULL,
  exon_features = NULL,
  ppi = NULL,
  source_data = c("rmats_hit", "hitindex", "rmats", "both"),
  event_types = c("ALE", "AFE", "MXE", "SE", "A3SS", "A5SS", "RI", "HFE", "HLE"),
  use = "JCEC",
  keep_annotated_first_last = FALSE,
  min_total_reads = 10L,
  minimum_proportion_containing_event = 0.5,
  terminal_fill = "event_max",
  cooks_cutoff = "Inf",
  adjust_method = "fdr",
  fdr_threshold = 0.05,
  delta_psi_threshold = 0.1,
  parallel_glm = TRUE,
  chunk_size_glm = 1000L,
  BPPARAM = BiocParallel::SerialParam(),
  chunk_size_match = 2000L,
  source_pairs = c("multi", "paired"),
  show_protein_domains = FALSE,
  return_class = c("data.table", "S4"),
  debug_steps = FALSE,
  metadata = list(),
  verbose = TRUE
)
```

Arguments

sample_frame	Optional sample manifest for [get_hitindex()] / [get_rmats_hit()]. Must include ‘path‘, ‘condition‘, and optional ‘sample_name‘.
data	Optional precomputed raw event-level table.
res	Optional precomputed differential inclusion table.
annotation_df	Optional annotation list from [get_annotation()] with ‘annotations‘ and ‘sequences‘.
protein_feature_total	Optional protein feature table from [get_comprehensive_annotations()]. Required if ‘exon_features‘ is not supplied and domain/PPI steps are run.
exon_features	Optional precomputed exon-feature overlap table from [get_exon_features()].
ppi	Optional preloaded PPI table. If ‘NULL‘, [get_ppi_interactions()] is used when needed.
source_data	Which ingestion route to use when ‘data‘ is ‘NULL‘. One of ‘"hitindex"‘, ‘"rmats"‘, ‘"both"‘, or legacy alias ‘"rmats_hit"‘.

`event_types` Event types for [get_rmats_hit()] / [load_rmats()]. Use both terminal and non-terminal types for ‘source_data = "both"‘.

`use` Junction count mode for rMATS ingestion (‘"JC"‘ or ‘"JCEC"‘).

`keep_annotated_first_last`
Passed to [get_hitindex()] for terminal events.

`min_total_reads`
Passed to [get_differential_inclusion()].

`minimum_proportion_containing_event`
Passed to [get_differential_inclusion()].

`terminal_fill` Passed to [get_differential_inclusion()].

`cooks_cutoff` Passed to [get_differential_inclusion()].

`adjust_method` Passed to [get_differential_inclusion()].

`fdr_threshold` Passed to [keep_sig_pairs()] as the adjusted p-value cutoff.

`delta_psi_threshold`
Passed to [keep_sig_pairs()] as the absolute delta-psi cutoff.

`parallel_glm` Passed to [get_differential_inclusion()].

`chunk_size_glm` Passed to [get_differential_inclusion()].

`BPPARAM` Passed to [get_differential_inclusion()]. Use a [BiocParallel::BiocParallelParam-class] object (for example [BiocParallel::SerialParam()], [BiocParallel::SnowParam()], or [BiocParallel::MulticoreParam()]).

`chunk_size_match`
Chunk size for [get_matched_events_chunked()].

`source_pairs` Pairing mode for [get_pairs()] (‘"multi"‘ or ‘"paired"‘).

`show_protein_domains`
Passed to [get_domains()].

`return_class` One of ‘"data.table"‘ or ‘"S4"‘.

`debug_steps` Logical; if ‘TRUE‘, includes intermediates (‘matched‘, ‘hits_sequences‘, ‘pairs‘, ‘seq_compare‘, ‘hits_domain‘) in data.table mode.

`metadata` Optional list attached to ‘SpliceImpactResult@metadata‘.

`verbose` Logical; emit progress messages.

Value

If ‘return_class = "data.table"‘, returns a named list with ‘data‘, ‘res‘, and ‘hits_final‘.

If ‘return_class = "S4"‘, returns a [SpliceImpactResult] containing ‘raw_events‘, ‘di_events‘, and ‘paired_hits‘ slots.

Examples

```
ex <- load_example_data(
  c("sample_frame", "annotation_df", "protein_feature_total", "ppi")
)
out <- get_splicing_impact(
  sample_frame = ex$sample_frame,
```

```
  annotation_df = ex$annotation_df,
  protein_feature_total = ex$protein_feature_total,
  ppi = ex$ppi,
  source_data = "rmats",
  event_types = c("SE"),
  use = "JCEC",
  parallel_glm = FALSE,
  BPPARAM = BiocParallel::SerialParam(),
  verbose = FALSE
)
print(names(out))
```

`get_user_data` *Get User-Supplied Splicing Event Data*

Description

SpliceImpactR accepts user-supplied splicing data in the same structure produced by `get_rmats_hit()`. Each event must be represented at the event-form and sample level.

Event representation - Each splicing event must be split into two forms: - `INC`: the inclusion isoform - `EXC`: the exclusion isoform - Alternative first/last exon events (AFE/ALE) or more abstract events may instead provide a single `SITE` form.

Coordinates - `inc` column: genomic coordinates included in the given form - `exc` column: genomic coordinates excluded in the given form - Coordinates may be one or multiple ranges (e.g., "100-200" or "100-150;300-350") - User must supply at least an inc and if supplying an exc, accompany with an inc coord

Counts and PSI - `inclusion_reads` and `exclusion_reads` must be provided per form - `psi` must be provided per sample (range 0-1) If psi isn't given, it will be extracted through `inclusion_reads/exclusion_reads`

Sample structure - Each event must have `INC` and `EXC` rows (or just `SITE`) - Each event must have >1 sample per condition (e.g., case vs control) - Required sample annotations: `sample`, `condition`

Required columns `event_id`, `event_type`, `form`, `gene_id`, `chr`, `strand`, `inc`, `exc`, `inclusion_reads`, `exclusion_reads`, `psi`, `sample`, `condition`

Defaults - If `event_type` not supplied: filled as `"unknown"` - If `source_file` not supplied: filled with empty string

This format enables downstream functionality including PSI modeling, annotation integration, and protein consequence prediction.

Usage

```
get_user_data(df)
```

Arguments

`df`	Data frame with splicing events. Detailed in description

Value

data.table with cols, detailed above: "event_id","event_type","form", "gene_id","chr", "strand", "inc","exc","inclusion_reads","exclusion_reads", "psi", "sample", "condition","source_file" – designed to match get_rmats_hit output

Examples

```
example_df <- data.frame(
  event_id = rep("A3SS:1", 8),
  event_type = "A3SS",
  form = rep(c("INC","EXC"), each = 4),
  gene_id = "ENSG00000158286",
  chr = "chrX",
  strand = "-",
  inc = c(rep("149608626-149608834",4), rep("149608626-149608829",4)),
  exc = c(rep("",4), rep("149608830-149608834",4)),
  inclusion_reads = c(30,32,29,31, 2,3,4,3),
  exclusion_reads = c(1,1,2,1, 28,27,26,30),
  sample = c("S1","S2","S3","S4","S1","S2","S3","S4"),
  condition = rep(c("case","case","control","control"), 2),
  stringsAsFactors = FALSE
)
example_df$psi < example_df$inclusion_reads / example_df$exclusion_reads
user_data <- get_user_data(example_df)
print(user_data)
```

---

get_user_data_post_di *Format user-supplied post-DI (post-differential-inclusion) splicing results*

Description

This function converts user-supplied event results into the internal SpliceImpactR DI format. It accepts per-event statistics and ensures each splicing event contains valid inclusion/exclusion structure.

Requirements **Event representation** - Each splicing event must be split into two forms: - `INC`: the inclusion isoform - `EXC`: the exclusion isoform - Alternative first/last exon events (AFE/ALE) or more abstract events may instead provide a single `SITE` form.

Coordinates - `inc` column: genomic coordinates included in the given form - `exc` column: genomic coordinates excluded in the given form - Coordinates may be one or multiple ranges (e.g., "100-200" or "100-150;300-350")

- User **must supply ‘event_id‘** (unique ID per splicing event, event_id = event_type:x for form != SITE. event_id = gene:event_type for form = SITE. Look at example output from get_differential_inclusion using test data for more examples) - Each event must be either: - **INC + EXC** forms (paired isoforms), or - **SITE** (single isoform) - Required columns: ‘gene_id, chr, strand, inc, exc, form, event_id‘

Behavior - Does **not** generate event IDs: user must provide them - Constructs ‘site_id = event_type|gene_id|chr|inc|exc|form‘ - ‘delta_psi‘: - If missing: INC = +1, EXC = -1, SITE expands to +1 and -1 each to get all relevant comparisons - ‘p.value‘, ‘padj‘: - If missing: set to 0 - All diagnostic fields (‘cooks_max, n, n_used, n_samples, n_case, mean_psi_ctrl, mean_psi_case, n_control‘) set to -1 if missing

Validation - Throws error if: - Any event lacks **INC+EXC** or **SITE** - An event mixes SITE with INC/EXC

Output Returns a ‘data.table‘ formatted like SpliceImpactR DI output. Ready for annotation, pairing, enrichment, and PPI analysis.

Usage

```
get_user_data_post_di(df)
```

Arguments

df	Data frame of post-DI results, detailed in description

Value

A data.table formatted like SpliceImpactR DI output (get_differential_inclusion)

Examples

```
example_user_data <- data.frame(
  event_id = rep("A3SS:1", 8),
  event_type = "A3SS",
  gene_id = "ENSG00000158286",
  chr = "chrX",
  strand = "-",
  form = rep(c("INC","EXC"), each = 4),
  inc = c(
    rep("149608626-149608834", 4),
    rep("149608626-149608829", 4)
  ),
  exc = c(
    rep("", 4),
    rep("149608830-149608834", 4)
  ),
  inclusion_reads = c(30, 28, 25, 32,  2, 3, 4, 3),
  exclusion_reads = c(1, 2, 1, 1, 28, 27, 26, 30),
  sample = c("S1","S2","S3","S4","S1","S2","S3","S4"),
  condition = rep(c("case","case","control","control"), 2),
  stringsAsFactors = FALSE
)

# compute psi if missing, just for demo

post_di_user_data <- get_user_data_post_di(example_user_data)
print(post_di_user_data)
```

`import_di_table` *Standardize a differential inclusion (DI) result table*

Description

Converts an arbitrary differential inclusion result table into the standardized column format expected by SpliceImpactR functions.

Usage

```
import_di_table(
  df,
 colmap = list(gene_id = "gene_id", chr = "chr", strand = "strand", inc = "inc", exc =
    "exc", delta_psi = "delta_psi", pvalue = "p.value", event_type = NULL),
  default_event_type = "SITE",
  adjust_method = "fdr",
  add_chr_prefix = FALSE
)
```

Arguments

`df`	A ‘data.frame‘ or ‘data.table‘ containing differential inclusion results.
`colmap`	A named list mapping required fields in ‘df‘ to standard names (‘gene_id‘, ‘chr‘, ‘strand‘, ‘inc‘, ‘exc‘, ‘delta_psi‘, ‘pvalue‘, and optionally ‘event_type‘).
`default_event_type`	Character. Default ‘event_type‘ to assign if none provided (default ‘"SITE"‘).
`adjust_method`	Character. Multiple-testing correction method passed to [stats::p.adjust()] (default ‘"fdr"‘).
`add_chr_prefix`	Logical. Add ‘"chr"‘ prefix if absent (default ‘FALSE‘).

Details

This function provides a uniform interface for importing external DI results (e.g. from rMATS, MAJIQ, or SUPPA2) so they can be compared or plotted alongside SpliceImpactR outputs.

Value

A standardized ‘data.table‘ with columns: ‘site_id‘, ‘event_type‘, ‘gene_id‘, ‘chr‘, ‘strand‘, ‘inc‘, ‘exc‘, ‘delta_psi‘, ‘p.value‘, ‘padj‘, and ‘form‘.

Examples

```
df <- data.frame(
  gene_id = "ENSG00000280071",
  chr = "7",
  strand = "+",
  inc = "1940088-1940549",
```

```
  exc = "",
  delta_psi = 0.25,
  p.value = 0.01
)
di_std <- import_di_table(df)
head(di_std)
```

`integrated_event_summary` *Integrated summary of event classification, alignment, and domain changes*

Description

Provides a comprehensive visualization and summary of event classification outcomes (frame shifts, rescues, matches, etc.), alignment quality, and domain change prevalence across alternative splicing event types. Integrates results from [‘compare_sequence_frame()‘] with pre-filter event tables to display pre- vs post-filter usage, event coordination, and gene/domain overlaps.

Usage

```
integrated_event_summary(hits, pre_filter_hits)
```

Arguments

`hits`	‘data.frame‘, ‘data.table‘, or ‘SpliceImpactResult‘ output from [‘compare_sequence_frame()‘], containing per-event alignment and domain information (e.g. ‘summary_classification‘, ‘prot_score‘, ‘event_type‘, ‘case_only_n‘, ‘control_only_n‘, etc.).
`pre_filter_hits`	‘data.frame‘, ‘data.table‘, or ‘SpliceImpactResult‘ representing the unfiltered input events (e.g. raw differential inclusion table prior to sequence comparison).

Details

The function integrates multiple layers of event-level characterization:

- Event-type composition and class proportions
- Protein alignment quality distribution
- Domain-change prevalence (case/control/both)
- Event-type coordination heatmap (Jaccard similarity across genes)
- Relative event retention pre- vs post-filtering

Value

A named list with:

'summaries' List containing per-type tables: `by_type`, `class_counts`, `score_summary`, `domain_prevalence`, and `relative_use`.

'plot' A multi-panel ['patchwork'] composite summarizing classification, alignment, domain changes, and coordination.

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
annotation_df <- load_example_data("annotation_df")$annotation_df
matched <- get_matched_events_chunked(res, annotation_df$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annotation_df$sequences)
pairs <- get_pairs(x_seq, source="multi")
seq_compare <-compare_sequence_frame(pairs, annotation_df$annotations)
annotation_df <- get_annotation(load = 'test')
interpro_features <- get_protein_features(c("interpro"), annotation_df$annotations, timeout = 600, test = TRUE)
protein_feature_total <- get_comprehensive_annotations(list(interpro_features))

exon_features <- get_exon_features(annotation_df$annotations, protein_feature_total)

hits_domain <- get_domains(seq_compare, exon_features)
ppi <- get_ppi_interactions()
hits_final <- get_ppi_switches(hits_domain, ppi, protein_feature_total)
int_summary <- integrated_event_summary(hits_final, res)
print(int_summary)
```

---

keep_sig_pairs *Filter event pairs by significance and deltaPSI thresholds*

Description

Keeps all rows belonging to events where at least one isoform or site passes adjusted p-value and deltaPSI significance criteria.

Usage

```
keep_sig_pairs(
  DT,
  padj_thr = 0.05,
  dpsi_thr = 0.1,
  return_class = c("auto", "data.table", "S4")
)
```

Arguments

DT	A ‘data.frame‘ or ‘data.table‘ containing at least ‘event_id‘, ‘padj‘, and ‘delta_psi‘ columns.
padj_thr	Numeric. Adjusted p-value threshold (default ‘0.05‘).
dpsi_thr	Numeric. Absolute deltaPSI threshold (default ‘0.1‘).
return_class	Character. Output mode: ‘"data.table"‘, ‘"S4"‘, or ‘"auto"‘ (default). In ‘auto‘, S4 input returns updated S4 output.

Value

A ‘data.table‘ (or updated ‘SpliceImpactResult‘ when ‘return_class‘ resolves to S4) containing all rows from event pairs in which at least one row meets the significance criteria.

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
sig_di <- keep_sig_pairs(res)
print(sig_di)
```

---

load_example_data *Load bundled example inputs for documentation*

Description

Returns commonly used example objects (sample manifest, test annotations, optional feature mappings, optional PPI table) so man-page examples can stay focused on the documented function rather than setup boilerplate.

Usage

```
load_example_data(
  include = c("sample_frame", "annotation_df"),
  biomaRt_databases = c("interpro"),
  test = TRUE
)
```

Arguments

include	Character vector selecting which objects to return. Supported values are ‘"sample_frame"‘, ‘"annotation_df"‘, ‘"protein_feature_total"‘, ‘"exon_features"‘, ‘"ppi"‘, and ‘"all"‘. If ‘"all"‘ is present, it expands to ‘c("sample_frame", "annotation_df", "protein_feature_total", "exon_features")‘.

biomaRt_databases
: Character vector passed to [get_protein_features()] when protein features are requested. Default is ‘"interpro"‘.

test
: Logical passed to [get_protein_features()] (default ‘TRUE‘).

Value

Named list containing the requested objects.

Examples

```
ex <- load_example_data(c("sample_frame", "annotation_df"))
sample_frame <- ex$sample_frame
annotation_df <- ex$annotation_df

ex2 <- load_example_data(c("annotation_df", "exon_features"))
exon_features <- ex2$exon_features
print(exon_features)
```

---

load_rmats *Load rMATS event files into standardized data.tables*

Description

Parses rMATS output (.MATS.JC.txt or .MATS.JCEC.txt) for multiple event types and returns unified event tables ready for downstream inclusion/exclusion processing.

Usage

```
load_rmats(
  paths,
  use = c("JC", "JCEC"),
  event_types = c("SE", "RI", "A5SS", "A3SS", "MXE")
)
```

Arguments

paths
: A data.frame with columns `path`, `sample_name`, and `condition`.

use
: Character scalar, one of `"JC"` or `"JCEC"`.

event_types
: Event types to include: one or more of `c("SE", "RI", "A5SS", "A3SS", "MXE")`.

Value

A ‘data.table‘ with unified rMATS event annotations, including columns:

- event_type, event_id, gene_id, chr, strand
- sample, condition (if applicable)
- delta_psi, pvalue, fdr (if present)
- inclusion/exclusion read counts

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
rmats <- load_rmats(sample_frame, use = "JCEC", event_types = c("MXE", "SE", "A3SS", "A5SS", "RI"))
print(rmats)
```

`overview_spicing_comparison` *Overview of global splicing event distributions between conditions*

Description

Generates a multi-panel comparison summarizing global splicing characteristics (event counts, events-per-gene, PSI distributions) between experimental and control conditions, normalized by sequencing depth.

Usage

```
overview_spicing_comparison(
  events,
  sample_df,
  depth_norm = c("exon_files", "user-given"),
  event_type = "AFE",
  conditions = c(control = "control", experimental = "case"),
  minReads = 10,
  output_file = NULL
)
```

Arguments

`events`	‘data.frame‘ or ‘data.table‘ of splicing events with at least: ‘sample‘, ‘condition‘, ‘gene_id‘, ‘inclusion_reads‘, ‘exclusion_reads‘, ‘psi‘, ‘inc‘, ‘exc‘.
`sample_df`	Metadata ‘data.frame‘ with sample-level paths and conditions.
`depth_norm`	Normalization mode: ‘’exon_files’‘ (compute from read files) or ‘’user-given’‘ (use provided size factors).
`event_type`	Character string specifying the event class (e.g., ‘"AFE"‘, ‘"ALE"‘).
`conditions`	Named character vector mapping ‘control‘ and ‘experimental‘ condition labels.
`minReads`	Minimum reads required for inclusion or exclusion (default = 10).
`output_file`	Optional path to save a combined summary figure.

Details

The output combines: - **Panel A:** Depth-normalized event counts per sample (Wilcoxon test) - **Panel B:** Mean events per gene per sample (Wilcoxon test) - **Panel D:** PSI cumulative distribution comparison (K-S test)

Size factors are computed internally using [.get_size_factors_from_exons()], enabling robust normalization.

Value

Invisibly returns a combined [‘patchwork‘] plot object.

See Also

[.get_size_factors_from_exons()], [getSizeFactors()]

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
ov <- overview_spicing_comparison(hit_index, sample_frame, 'exon_files')
print(ov)
```

---

plot_alignment_summary

Plot alignment score distribution and coding summary

Description

Visualizes alignment scores (‘prot_pid‘ or ‘dna_pid‘) and coding class composition of hits, showing both the categorical composition and histogram of alignment identity values.

Usage

```
plot_alignment_summary(
  hits,
  mode = c("protein", "transcript"),
  output_file = NULL
)
```

Arguments

hits	‘data.frame‘ or ‘data.table‘ containing ‘prot_pid‘ or ‘dna_pid‘ and ‘summary_classification‘ columns.
mode	Character, either ‘"protein"‘ (uses ‘prot_pid‘) or ‘"transcript"‘ (uses ‘dna_pid‘).
output_file	Optional path to save the combined plot.

Value

A composite ‘ggplot‘ object (from ‘ggpubr::ggarrange‘) showing stacked bar counts by coding class and histogram of alignment identity scores.

See Also

[plot_length_comparison()]

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
annotation_df <- load_example_data("annotation_df")$annotation_df
matched <- get_matched_events_chunked(res, annotation_df$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annotation_df$sequences)
pairs <- get_pairs(x_seq, source="multi")
seq_compare <-compare_sequence_frame(pairs, annotation_df$annotations)
alignment_summary <- plot_alignment_summary(seq_compare)
print(alignment_summary)
```

---

plot_di_volcano_dt *Volcano plot for differential inclusion results*

Description

Generates a volcano plot of deltaPSI vs. -log10(FDR) highlighting significant events.

Usage

```
plot_di_volcano_dt(di, padj_thr = 0.05, dpsi_thr = 0.1)
```

Arguments

di	A ‘data.frame‘ or ‘data.table‘ containing at least ‘delta_psi‘ and ‘padj‘ columns (optionally ‘event_type‘).
padj_thr	Numeric. Adjusted p-value threshold (default ‘0.05‘).
dpsi_thr	Numeric. Absolute deltaPSI threshold (default ‘0.1‘).

Details

Significant sites are colored in ‘deeppink4‘; nonsignificant sites are shown in light grey. Dashed and dotted lines indicate deltaPSI and FDR thresholds.

Value

A ‘ggplot2‘ object showing differential inclusion significance.

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
plot_di_volcano_dt(res)
```

`plot_enriched_domains_counts` *Plot enriched domains by associated event count*

Description

Visualizes the top enriched protein domains based on the number of events contributing to each domain's enrichment, optionally coloring by -log10 adjusted p-value.

Usage

```
plot_enriched_domains_counts(enriched_domains, top_n = 25)
```

Arguments

`enriched_domains`	'data.frame' or 'data.table' Output table from [enrich_domains_hypergeo()], including at least columns 'domain_id' and 'events'. Optional columns 'padj' and 'OR' are used for coloring and labeling.
`top_n`	Integer (default '25') Number of top domains to display, ranked by increasing 'padj'.

Details

The function expects the output of [enrich_domains_hypergeo()], typically a 'data.table' or 'data.frame' containing 'domain_id', 'events', and optionally 'padj' and 'OR'.

Each bar corresponds to one domain, with height proportional to the number of unique 'event_id's contributing to that domain. Bars are ordered by ascending adjusted p-value ('padj'), and colored by -log10(padj) if available. When no 'padj' column is present, the bars are shown in a uniform fill color.

Value

A 'ggplot' object showing bars of domain counts colored by enrichment significance.

See Also

[enrich_domains_hypergeo()], [enrich_by_event()], [enrich_by_db()]

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
annotation_df <- load_example_data("annotation_df")$annotation_df
matched <- get_matched_events_chunked(res, annotation_df$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annotation_df$sequences)
pairs <- get_pairs(x_seq, source="multi")
seq_compare <-compare_sequence_frame(pairs, annotation_df$annotations)
annotation_df <- get_annotation(load = 'test')
interpro_features <- get_protein_features(c("interpro"), annotation_df$annotations, timeout = 600, test = TRUE)
protein_feature_total <- get_comprehensive_annotations(list(interpro_features))

exon_features <- get_exon_features(annotation_df$annotations, protein_feature_total)

hits_domain <- get_domains(seq_compare, exon_features)
bg <- get_background(source = "hit_index",
                     input = sample_frame,
                     annotations = annotation_df$annotations,
                     protein_features = protein_feature_total)
enriched_domains <- enrich_domains_hypergeo(hits_domain, bg, db_filter = 'interpro')
plot_enriched_domains_counts(enriched_domains, top_n = 20)
```

---

plot_length_comparison

Compare inclusion vs. exclusion isoform lengths

Description

Generates a multi-panel summary comparing isoform lengths between inclusion (INC) and exclusion (EXC) events for either protein or transcript modes.

Usage

```
plot_length_comparison(
  hits,
  phenotypes = c(control = "control", experimental = "case"),
  mode = c("protein", "transcript"),
  output_file = NULL
)
```

Arguments

hits	‘data.frame‘ or ‘data.table‘ containing event-level results, including at least ‘prot_len_case‘, ‘prot_len_control‘, and ‘prot_len_diff‘ for ‘mode = "protein"‘, or their transcript analogues ‘tx_len_case‘, ‘tx_len_control‘, and ‘tx_len_diff‘.

phenotypes	Named character vector of length 2 giving labels for control and experimental phenotypes. Must have names ‘"control"‘ and ‘"experimental"‘.
mode	Character, one of ‘"protein"‘ or ‘"transcript"‘. Determines which length columns to use and whether to derive protein-coding categories.
output_file	Optional path for saving the combined plot (e.g., ‘"length_summary.png"‘).

Details

Produces three coordinated panels:

1. Paired boxplot showing INC vs EXC lengths per event with Wilcoxon paired test annotation.
2. Density plot of delta length (INC - EXC).
3. Barplot summarizing protein-coding categories.

Value

A composite ‘ggplot‘ object (assembled with ‘patchwork‘) showing paired boxplots, delta-length density, and protein-coding class distribution.

See Also

[plot_alignment_summary()]

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
annotation_df <- load_example_data("annotation_df")$annotation_df
matched <- get_matched_events_chunked(res, annotation_df$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annotation_df$sequences)
pairs <- get_pairs(x_seq, source="multi")
seq_compare <-compare_sequence_frame(pairs, annotation_df$annotations)
proximal_output <- plot_length_comparison(seq_compare)
print(proximal_output)
```

---

plot_ppi_summary *Plot summary of altered PPI interactions*

Description

Visualizes the frequency and magnitude of gained/lost PPIs per event, using a dual-panel layout: - left: proportion of events with any PPI change - right: histograms of CASE and CONTROL partner counts (non-zero only)

Usage

```
plot_ppi_summary(
  df,
  bins = 30,
 palette = c(no = "grey80", yes = "deeppink4", CASE = "#2b8cbe", CONTROL = "#e34a33"),
  output_file = NULL,
  width = 9,
  height = 4.8
)
```

Arguments

df	‘data.table‘ or ‘data.frame‘ with PPI counts per event, as returned by [ppi_switches_for_hits()].
bins	Integer; number of histogram bins (default ‘30‘).
palette	Named character vector of fill colors for the plot (default includes ‘"no"‘, ‘"yes"‘, ‘"CASE"‘, ‘"CONTROL"‘).
output_file	Optional path to save the figure (‘.png‘ or ‘.pdf‘).
width, height	Numeric dimensions (in inches) for saved plot.

Value

A ‘ggplot‘ object combining two panels (using ‘patchwork‘).

See Also

[ppi_switches_for_hits()]

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
annotation_df <- load_example_data("annotation_df")$annotation_df
matched <- get_matched_events_chunked(res, annotation_df$annotations, chunk_size = 2000)
x_seq <- attach_sequences(matched, annotation_df$sequences)
pairs <- get_pairs(x_seq, source="multi")
seq_compare <-compare_sequence_frame(pairs, annotation_df$annotations)

annotation_df <- get_annotation(load = 'test')
interpro_features <- get_protein_features(c("interpro"), annotation_df$annotations, timeout = 600, test = TRUE)
protein_feature_total <- get_comprehensive_annotations(list(interpro_features))

exon_features <- get_exon_features(annotation_df$annotations, protein_feature_total)

hits_domain <- get_domains(seq_compare, exon_features)

bg <- get_background(source = "hit_index",
                     input = sample_frame,
```

```
                    annotations = annotation_df$annotations,
                    protein_features = protein_feature_total)
ppi <- get_ppi_interactions()
hits_final <- get_ppi_switches(hits_domain, ppi, protein_feature_total)
ppi_plot <- plot_ppi_summary(hits_final)
print(ppi_plot)
```

`plot_two_transcripts_with_domains_unified` *Plot two transcripts with exon structure and protein feature tracks (unified view)*

Description

Visualize **two Ensembl transcripts** side-by-side with their exon structures and optional **protein feature/domain tracks** (e.g., InterPro, Pfam, ELM, SEG, SignalP), using a single entrypoint. The plot can be rendered in either:

Usage

```
plot_two_transcripts_with_domains_unified(
  ...,
  view = c("transcript", "protein")
)
```

Arguments

`...` Additional arguments forwarded to the internal workhorse `plot_two_transcripts_with_features()`. Common arguments include:

transcripts Character vector of length 2 of Ensembl transcript IDs.

gtf_df GTF-like exon annotation table containing at least `transcript_id`, `type=="exon"`, `exon_number`, `chr`, `strand`, `start`, `end`.

protein_features Protein feature table with at least `ensembl_transcript_id`, `name`, `feature_id`, `database`.

feature_db Optional character vector of databases to retain (e.g., `c("interpro","pfam","elm","seg"`,

wrap_width Integer; width for wrapping long domain labels.

highlight_hits Optional data.frame/data.table of event rows used for highlighting.

highlight_event_id Optional event ID to select from `highlight_hits`.

highlight_alpha Alpha transparency for highlight bands.

highlight_box Logical; draw dashed vertical bounds around highlighted spans.

highlight_box_pad_frac Fraction of total x-range used to pad highlight bounds.

highlight_box_lwd Line width for highlight bounding lines.

combine_domains Logical; combine identical domain labels onto shared tracks.

domain_base_gap Vertical gap between transcript backbone and first domain track.

domain_track_step Vertical spacing between stacked domain tracks.

domain_label_dy Vertical offset used to place domain labels.

`view` Character scalar selecting the visualization coordinate system: `"transcript"` (genomic/intron-aware) or `"protein"` (compact/exonic).

Details

- **Transcript view** (‘view = "transcript"‘): genomic x-axis with introns drawn and, when both transcripts are negative-strand, a strand-aware x-axis reversal so the display reads left-to-right in **5’→3’** direction. - **Protein view** (‘view = "protein"‘): compact, intron-free x-axis where exons are concatenated end-to-end (exonic coordinates), making it easier to compare protein feature locations across isoforms without large genomic intron gaps.

Optionally, event-specific genomic spans (e.g., inclusion/exclusion regions) can be overlaid as translucent highlight bands per transcript.

What gets drawn

- Exons as black rectangles (alternating alpha in protein/compact view).
- Introns as grey connector segments (transcript/genomic view only).
- Protein features as stacked rectangles under each transcript (if present).
- Domain labels anchored to the left edge (or right when the x-axis is reversed).
- Optional highlighted spans with dashed bounding lines.

Protein/domain tracks Protein features are filtered to the two transcripts and optionally filtered by `feature_db`. Features are clipped to exons in transcript/genomic view, and projected into compact/exonic coordinates in protein view.

- If `combine_domains = TRUE`, identical domain labels share a common vertical track to reduce redundancy.
- If `combine_domains = FALSE`, each feature instance is assigned its own track (potentially more vertical space, but preserves instance-level separation).

Event highlighting If `highlight_hits` and `highlight_event_id` are provided, event spans are parsed from `inc_case`, `exc_case`, `inc_control`, `exc_control` columns. In transcript/genomic view spans are used directly; in protein/compact view spans are projected into compact coordinates per transcript using exon maps.

Value

A `ggplot` object, or `NULL` if no drawable content is available (e.g., when both transcripts have zero features and you have configured internal logic to early-return on missing domains).

Expected input formats

protein_features `name` **column:** The internal parser expects feature genomic coordinates encoded in the `name` field formatted as `"<label>;chr:start-end"` (e.g. `"PF00069;chr7:123-456"`).

highlight_hits span columns: Span columns are expected as strings `"start-end"` using genomic coordinates. Missing spans should be `NA_character_`.

Highlight input (custom_hits_domain) example

The highlight_hits object is expected to be a table (data.frame/data.table) with at least one row per event. Use highlight_event_id to select the row to plot. Required columns are event_id, event_type_control, transcript_id_case, transcript_id_control. Span columns may include inc_case, exc_case, inc_control, exc_control and should be genomic coordinate strings of the form "start-end" (or NA_character_ when absent).

```
custom_hits_domain <- data.table::data.table(
  event_id = event:n,
  event_type = event,
  transcript_id_case = transcript_id,
  transcript_id_control = transcript_id,
  inc_case = inc_case,
  inc_control = inc_control,
  exc_case = exc_case,
  exc_control = exc_control
)
```

See Also

ggplot for rendering and theming.

Examples

```
# Example highlight row (skipped exon / SE), but can usually just use
# hits_domain/hits_final and supply the event_id in highlight_event_id
custom_hits_domain <- data.table::data.table(
  event_id = "AFE:1",
  event_type = "AFE",
  transcript_id_case = "ENST00000337907",
  transcript_id_control = "ENST00000476556",
  inc_case = "8655973-8656441",
  inc_control = "8423561-8423666",
  exc_case = NA,
  exc_control = NA
)

# Transcript (genomic) view: introns included, strand-aware axis
p_tx <- plot_two_transcripts_with_domains_unified(
  gtf_df = annotation_df$annotations,
  protein_features = protein_feature_total,
  feature_db = c("interpro", "pfam"),
  highlight_hits = custom_hits_domain,
  highlight_event_id = "AFE:1",
  combine_domains = FALSE,
  view = "protein"
)

# Protein (compact) view: introns removed
p_prot <- plot_two_transcripts_with_domains_unified(
  gtf_df = annotation_df$annotations,
```

```
  protein_features = protein_feature_total,
  feature_db = c("interpro", "pfam", "elm", "seg"),
  highlight_hits = hits_final,
  highlight_event_id = "ENSG00000142599:AFE",
  combine_domains = TRUE,
  view = "transcript"
)

# We are also able to just probe 2 random transcripts from annotations
p_prot <- plot_two_transcripts_with_domains_unified(
  transcripts = c("ENST00000337907","ENST00000476556"),
  gtf_df = annotation_df$annotations,
  protein_features = protein_feature_total,
  feature_db = c("interpro", "pfam", "elm", "seg"),
  combine_domains = TRUE,
  view = "protein"
)
```

`probe_individual_event` *Visualize PSI values for a single splicing event*

Description

Creates a per-event PSI summary across samples. For **AFE** and **ALE** events, the plot separates PSI by ‘inc‘ entry to distinguish alternative terminal exons; other event types are summarized by ‘event_id‘.

Usage

```
probe_individual_event(data, event, fill_zeros = TRUE)
```

Arguments

`data`	‘data.frame‘ or ‘data.table‘ containing at least the columns ‘event_id‘, ‘event_type‘, ‘psi‘, ‘condition‘, and ‘sample‘. For terminal events, an ‘inc‘ column is also required.
`event`	Character scalar specifying the ‘event_id‘ to visualize.
`fill_zeros`	Logical indicating whether to fill missing PSI values with zeros for samples that contain any observation of the event (default: ‘TRUE‘).

Value

A list with elements:

- ‘plot‘: ‘ggplot‘ box/point plot of PSI per sample group.
- ‘data‘: ‘data.table‘ used to build the plot.

Examples

```
ex <- load_example_data("sample_frame")
sample_frame <- ex$sample_frame
hit_index <- get_hitindex(sample_frame)
res <- get_differential_inclusion(hit_index)
event_probe <- "ENSG00000117632:AFE"
probe_individual_event(hit_index, event = event_probe)
```

---

spliceimpact_s4_guide *Detailed guide for using SpliceImpactResult*

Description

Prints practical guidance on slots, assays, key columns, and common access patterns for conversion between S4 and ‘data.table‘.

Usage

```
spliceimpact_s4_guide(as_markdown = FALSE)
```

Arguments

as_markdown	Logical; if ‘TRUE‘, returns guide text instead of printing.

Value

Invisible character guide text (or visible text if ‘as_markdown = TRUE‘).

Examples

```
guide_txt <- spliceimpact_s4_guide(as_markdown = TRUE)
cat(substr(guide_txt, 1, 80), "\n")
```

---

spliceimpact_s4_schema *S4 slot and key schema for SpliceImpactResult*

Description

S4 slot and key schema for SpliceImpactResult

Usage

```
spliceimpact_s4_schema()
```

Value

Named list describing slots, core key columns, and assay names.

Examples

```
schema <- spliceimpact_s4_schema()
names(schema)
```

`SpliceImpactResult-class`

SpliceImpact result container (S4)

Description

SpliceImpact result container (S4)

Slots

`raw_events` 'SummarizedExperiment' of sample/form-level rows.

`di_events` 'GRanges' of differential inclusion rows.

`res_di` 'GRanges' of threshold-filtered differential rows.

`matched` 'S4Vectors::DataFrame' of annotation-matched DI rows.

`sample_frame` 'S4Vectors::DataFrame' sample manifest ('path', 'sample_name', 'condition') when available.

`paired_hits` 'GRanges' of paired event-level rows.

`segments` 'GRangesList' of per-event segment parts ('inc_case', etc.).

`metadata` list with flags and optional provenance.

Index